MÁSTER

**MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS**

CÓDIGO 31105151- TRABAJO FIN DE MÁSTER

# Generación automática de código, documentación y datos de prueba unitarias para entornos productivos Mainframes


**|ALUMNO. Abel García Sánchez**
**|DIRECTOR. Rubén Heradio Gil**


**2018/2019 - JUNIO**

UNED – E.T.S DE INGENIERÍA INFORMÁTICA

**MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS**

CÓDIGO 31105151- TRABAJO FIN DE MÁSTER


# Generación automática de código, documentación y datos de prueba unitarias para entornos productivos Mainframes

Trabajo Fin de Máster Tipo B

**|ALUMNO. Abel García Sánchez**
**|DIRECTOR. Rubén Heradio Gil**

## DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 19/06/2019

Quién suscribe:

Autor(a): ABEL GARCÍA SÁNCHEZ
D.N.I/N.I.E/Pasaporte.: 53133187J

Hace constar que es la autor(a) del trabajo:

Título completo del trabajo.

**Generación automática de código, documentación y datos de prueba unitarias para entornos productivos Mainframes**

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

**DECLARACIÓN:**

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

IMPRESO TFDM05_AUTORPBL
AUTORIZACIÓN DE PUBLICACIÓN
CON FINES ACADÉMICOS

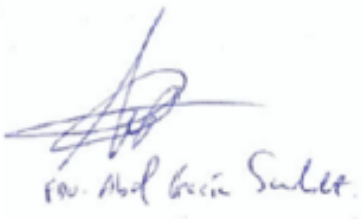
UNED ETS de Ingeniería Informática

**Impreso TFdM05_AutorPbl. Autorización de publicación y difusión del TFM para fines académicos**

## Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es



# RESUMEN

El presente proyecto tiene como objetivo el desarrollo de una herramienta para la generación automática de código y de documentación, la cual tendrá implícitas unas funcionalidades que servirán de facilitadores para sistemas Mainframes.

El desarrollo estará apoyado por técnicas GAC (generación automática de código) y SPL (líneas de productos de software), tales como MDD, MDA, inyección de código, uso de plantillas, etc.

El proyecto abordará los siguientes puntos:

- Situación actual, evolución de los sistemas Mainframes y problemática actual.
- Análisis e investigación de herramientas facilitadoras existentes en el mercado para las diferentes funcionalidades.
- Análisis, diseño y desarrollo de la herramienta. Las funcionalidades principales de esta herramienta serán las siguientes:
    - Generación automática de documentación técnica. Partiendo del código JCL (Job Control Language) se generará de forma automática un documento con información técnica sobre la arquitectura de componentes Software (elementos software, tipo, nombre, etc), un diagrama de DFD de ejecución con un nivel de abstracción 1, otro DFD detallado con los componentes del código y el detalle de cada uno de los pasos que forman el JCL, indicando interfaces de E/S y descripción del paso.
    - Generación automática de documentación técnica, generación automática de código y análisis de código COBOL. Partiendo del código fuente de un programa COBOL se generará un entorno de pruebas generando el JCL correspondiente, un programa COBOL principal llamador y un documento técnico que incluirá la arquitectura de componentes, análisis de complejidad y calidad, grafo de caminos, detección de errores potenciales, etc.
- Validación experimental. Realización de pruebas íntegras de la herramienta para validar todas sus funcionalidades.
- Conclusiones y trabajo futuro sobre el proyecto desarrollado.

Palabras clave: GAC (Generación Automática de Código), Mainframes, MDD (*Model-driven development)*, MDA (Model-Driven Architecture), pruebas unitarias, JCL, COBOL, generadores, arquitectura, metamodelo, abstracción, pruebas.

# CONTENIDO

# Control de versiones

| Versión | Fecha | Descripción de los cambios |
|---|---|---|
| 1.0 | 10/03/2019 | Borrador |
| 1.1 | 20/05/2019 | Primera entrega del trabajo |
| 2.0 | 06/06/2019 | Segunda entrega con actualización según indicaciones del tutor |
| 2.1 | 13/06/2019 | Tercera entreca con ajustes según indicados por el tutor. |
| 2.2 | 18/06/2019 | Inclusión de autorizaciones. |
| 2.3 | 21/06/2019 | Ajuste de formatos |

# Índice de figuras

# Índice de tablas

# CAPÍTULO 1

# 1. Introducción

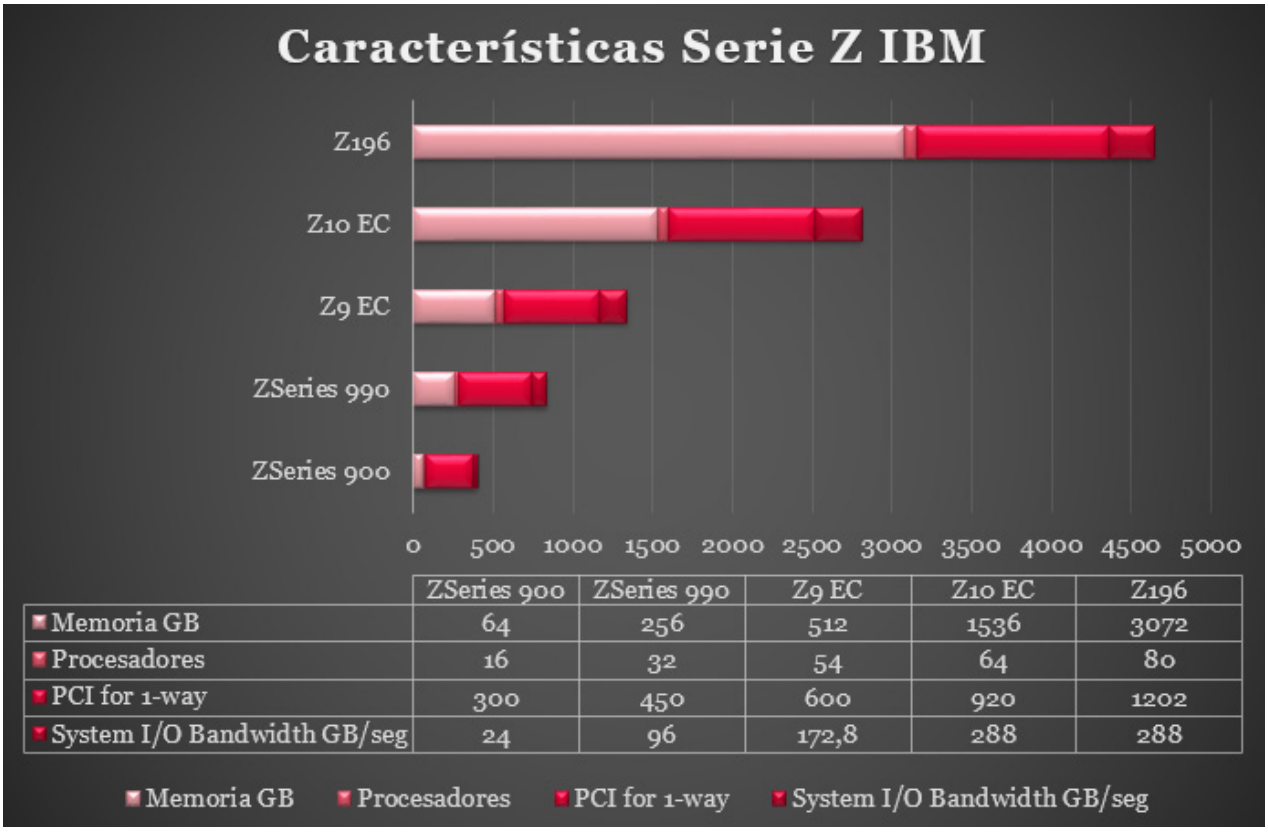
En este capítulo se analizará la evolución de los sistemas Mainframes, el cambio de tendencia en el uso de tecnologías, la problemática generada por el cambio de tendencia y por útlimo la arquitectura planteada para dar solución al problema descrito.

## *1.1 Evolución tecnológica de los sistemas Mainframes*

Si echamos la vista atrás, en las primeras décadas del siglo pasado, con la aparición de los grandes computadores como el ENIAC (considerado por muchos como el primer computador digital en 1946), podremos observar la gran evolución tecnológica que ha experimentado la humanidad, cómo ha ido aumentado la capacidad de cálculo, la capacidad de almacenamiento de cientos o miles de bytes a cientos de Terabyte, y la miniaturización que han ido experimentando los aparatos electrónicos, desde esos primeros ordenadores fabricados a base de válvulas de vacío, que ocupaban edificios enteros, hasta los actuales servidores compuestos de circuitos integrados fabricados con las últimas técnicas de miniaturización y de nanotecnología

En nuestro tema de estudio, nos centraremos en los entornos Mainframes que es un término utilizado para referirse a grandes equipos centralizados, que son utilizados por grandes compañías en sus Centro de Proceso de Datos (CPD), como por ejemplo los IBM de la serie Z y la problemática existente para el desarrollo y mantenimiento de software en esta tipología de entornos.

En el anexo '**Evolución de Mainframes IBM'** [1] se realiza un análisis sobre la evolución que ha experimentado este tipo de máquina desde su primera generación en 1952 (IBM 701) hasta la actualidad con sus modelos Z, que ya cuentan con una capacidad de almacenamiento y de cálculo muy avanzados. A continuación, en la figura 1, se detallan las características de la última serie de Mainframes, la serie Z.




| | ZSeries 900 | ZSeries 990 | Z9 EC | Z10 EC | Z196 |
|---|---|---|---|---|---|
| Memoria GB | 64 | 256 | 512 | 1536 | 3072 |
| Procesadores | 16 | 32 | 54 | 64 | 80 |
| PCI for 1-way | 300 | 450 | 600 | 920 | 1202 |
| System I/O Bandwidth GB/seg | 24 | 96 | 172,8 | 288 | 288 |

*Figura 1. Características de computación MODELOS Z*

## *1.2 Cambio de tendencia*

Desde los años 70 hasta principios de los 90, los Mainframes fueron los que soportaron el 90% de las operaciones de las grandes compañías, desde las transacciones on-line, con el soporte de la arquitectura front, hasta los procesos nocturnos batch, pero con el desarrollo de nuevas formas de computación, programación orientada a objetos y otras tecnologías no soportadas en estos sistemas, gran parte de los procesos han sido trasladados a otro tipo de máquinas con otras arquitecturas.

Se pueden diferenciar 3 grandes etapas:

Etapa I. Décadas 70 y 80.

Básicamente existía una sola capa de software en los entornos productivos, aunque funcionalmente se podía dividir en 3, tal como se puede observar en la figura 2:

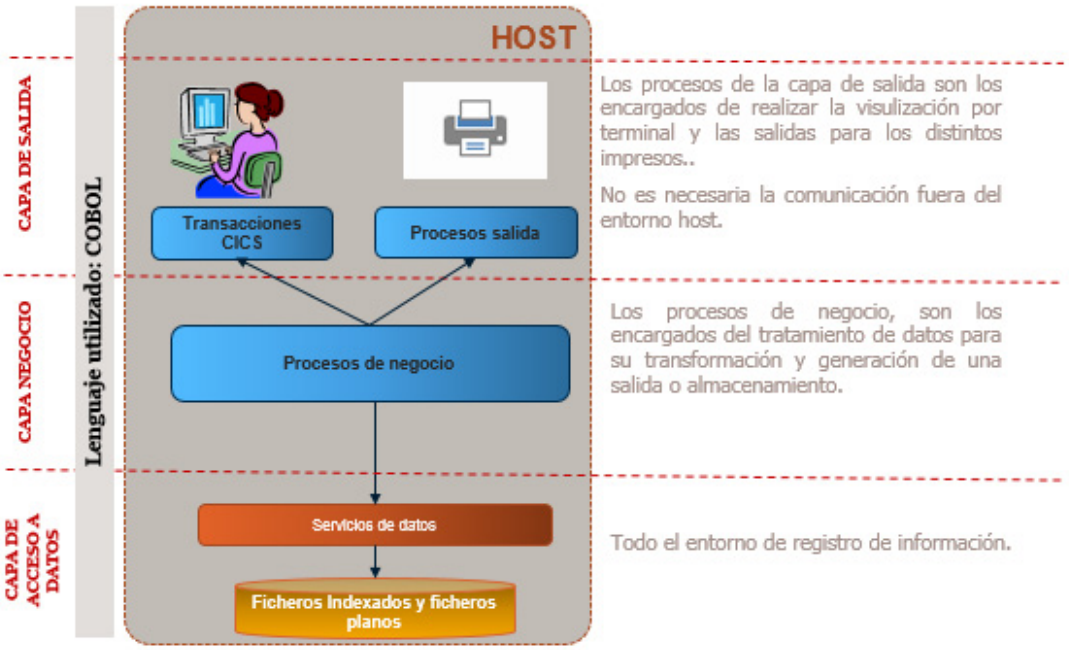



*Figura 2. Arquitectura etapa I (décadas 70-80)*

Tal como se puede observar en la anterior gráfica, todo estaba programado en Cobol, desde las pantallas de visualización para el usuario, hasta las salidas de impresos, pasando por procesos de transformación y tratamiento de datos.

En esta etapa, la información se solía registrar en ficheros planos o en ficheros VSAM (ficheros indexados), todavía no estaba extendida la utilización de bases de datos, aunque ya a finales de los 80 empezó su despliegue, pero no sería hasta principio de los 90 cuando las BD tuvieron su mayor expansión.

En esta etapa, para el entorno de visualización por pantalla se utilizaba el entorno transaccional de CICS para poder visualizar y recibir la información de los mapas (mapas era la denominación que tenían las pantallas) se utilizaban las sentencias de SEND y RECEIVE del entorno CICS. Ejemplo:

```
    EXEC CICS
         SEND FROM(MAPA-EJEMPLO)
         LENGTH (LENGTH OF MAPA-FECHA)
         ERASE
      END-EXEC
      EXEC CICS
         RECEIVE
         INTO (MAPA-EJMPLO)
         LENGTH(LENGTH OF MAPA-FECHA) NOHANDLE
       END-EXEC
```

Etapa II. Décadas 90.

En esta década, se inició el desarrollo de nuevas API presentación para el usuario final, y se incluyó una nueva FRONT para este tipo de desarrollos, la cual ya no estaba escrita en lenguaje COBOL y tampoco se encontraba ubicada en sistemas MAINFRAMES. La arquitectura utilizada se puede observar en la figura 3:

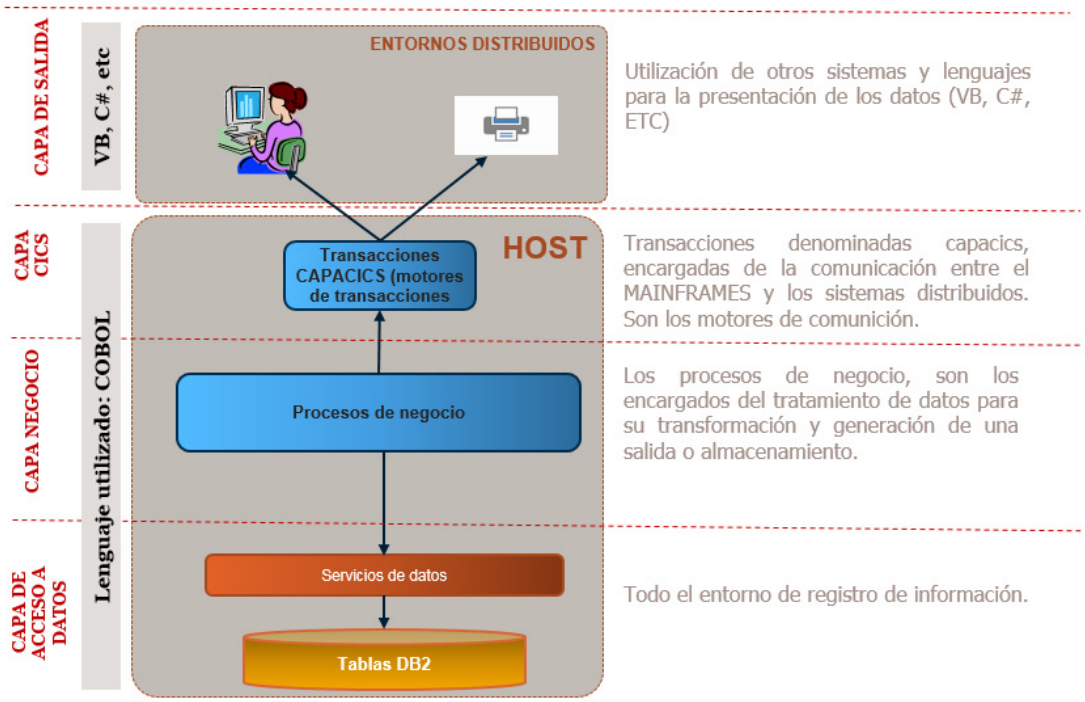



*Figura 3. ARQUITECTURA ETAPA II (década 90)*

En este tipo de arquitecturas, los programas externos al sistema centralizado ya no estaban escritos en COBOL sino en otros lenguajes más adaptables a los sistemas distribuidos.

Otro cambio significativo fue la sustitución de los ficheros indexados por tablas DB2 para la gestión de la información.

Etapa III. Actualidad.

Se incluyen nuevas capas sustituyendo los motores transaccionales (los CAPACICS) escritos en COBOL por modernos motores que establecerán comunicaciones con diferentes entornos distribuidos y servicios WEB, tal como se puede observar en la figura 4:

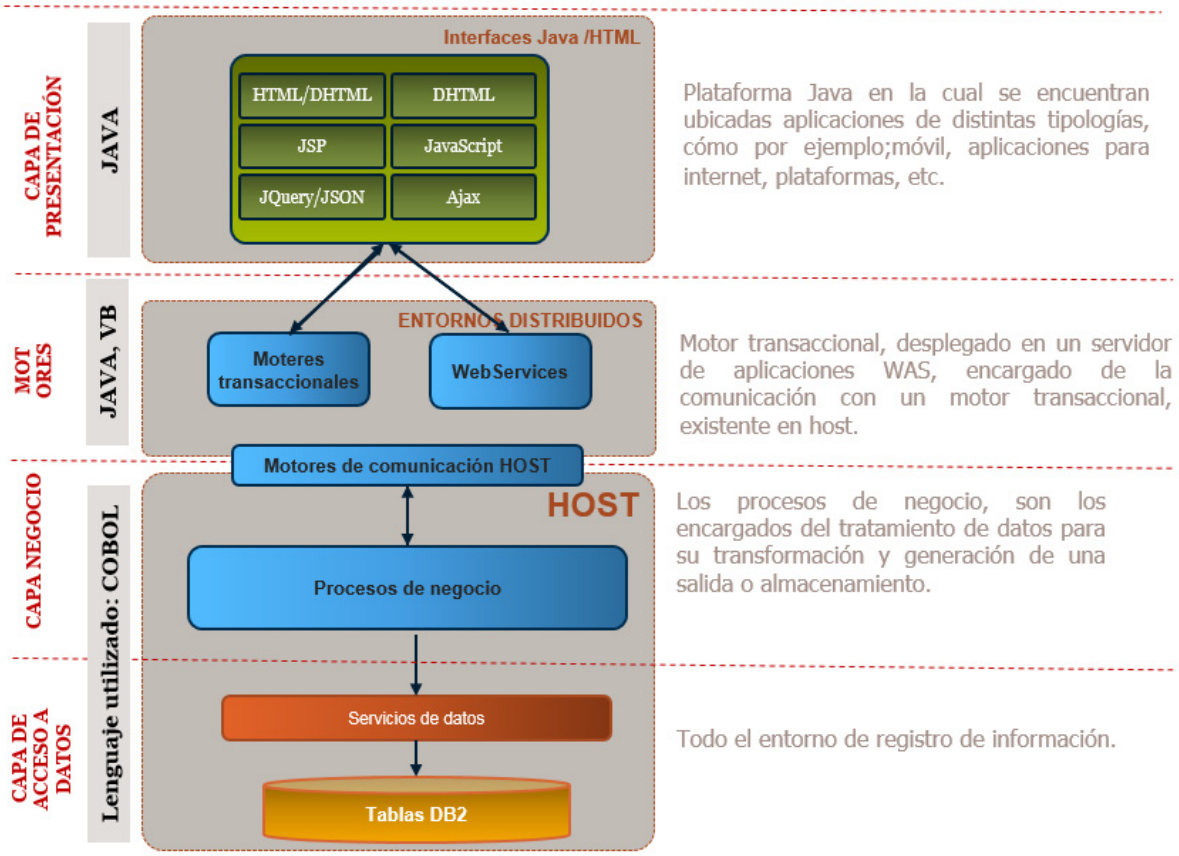



*Figura 4. Arquitectura etapa III (actualidad)*

En esta última etapa, el núcleo del negocio y el acceso a la información sigue estando programado en COBOL, por lo que sigue existiendo un gran mercado para este lenguaje de programación, no solo para su mantenimiento, sino para nuevos evolutivos que incidan en la capa de negocio.

## *1.3 Problemática y necesidad de herramientas para estos sistemas*

Hoy en día (año 2019), estos sistemas todavía soportan gran cantidad de procesos, los cuales tendrían un elevado coste si se decidiera migrar a otro tipo de sistemas, por lo que los Mainframes siguen y seguirán siendo una parte fundamental en las grandes compañías a corto y medio plazo tal como siguen publicando estudios realizados y análisis en artículos de revistas [2] [3].

Las tipologías de procesos [4] más importantes dentro de estos sistemas son los siguientes:

- Acceso a base de datos. El SGBD generalizado en estos sistemas es el DB2. Un porcentaje muy alto de información está guardado en esta SGBD, y los accesos a esta información se realizan a través del lenguaje SQL que se ubica en programas COBOL mediante referencias EXEC …. END-EXEC.

- Procesos batch. Son procesos (por norma general nocturnos), en los cuales se realizan operaciones sobre información recogida en el on-line del día (arquitectura front). Estos procesos tratan gran cantidad de información y pueden ir desde liquidaciones de cuentas, transferencias, hasta pago de impuestos a la AEAT.

- Transaccionales. Procesos on-line que dan soporte a otros sistemas externos, generalmente de comunicación para dar soporte a entorno FRONT, ya sea sobre plataformas monopuesto o web. Estos sistemas transaccionales están soportados por CICS.

Las tecnologías principales utilizadas en estos sistemas son:

- Lenguajes de programación utilizados. Principalmente el COBOL. Todavía puede existir algún programa en aplicaciones muy antiguas escritos en Assembler

- Sistema operativo. MVS. —Z/OS Tiene más facilidades que el resto, trabajaban con terminales actualmente se usan los emuladores para simular las antiguas “pantallas negras” tal como el 3270.

- SGBD. La principal es la BD estructural DB2, pero también existe la Bases de Datos IMS (DL/1), aunque en extinción.

- Sistema transaccional. Utilización de CICS.

- Procesos batch. Para la configuración de los entornos batch se utiliza el JCL.

Una de las principales problemáticas que se encuentra en estos sistemas, es la falta de formación que los nuevos graduados (a través de cursos, módulos o ingenierías) tienen sobre este tipo de tecnologías.

Esta falta de formación hace complicado el mantenimiento de aplicaciones ya consolidadas en estos sistemas, tanto para pequeñas modificaciones en programas COBOL, como para preparar los datos y el entorno de las pruebas unitarias.

Otra problemática, es la obsolescencia del software y la falta de documentación o su desactualización. Esto provoca tener gran cantidad de software y no tener una visión global de su funcionamiento.

Podríamos definir 4 facilitadores tecnológicos principales que ‘faciliten’ el desarrollo de proyectos o convivencias con arquitecturas Mainframes. Cada uno de estos facilitadores incorpora factores de gran importancia para alcanzar los beneficios esperados de este tipo de arquitecturas.


*Figura 5. Propuesta de facilitadores tecnológicos Mainframes*

Descripción de los facilitadores definidos en la figura 5 (*):

- GAD para procesos JCL. Generación Automática de Documentación a partir del código de procesos JCL. Con este facilitador, lo que se pretende es mostrar el funcionamiento y utilidad de un proceso JCL a personas que desconocen este lenguaje y son ajenas a las arquitecturas Mainframes. También es una herramienta que ayuda a documentar aplicaciones no documentadas.

- Analizador programas COBOL. Este facilitador tendrá las siguientes funciones:
  - Preparación entorno de pruebas unitarias del programa, generando el código del programa principal necesario y el JCL.
  - Analizar el nivel de calidad del código.
  - Detección de potenciales errores en la codificación.
  - Generación automática de la documentación técnica del programa.

- GAC programas acceso a SGBD. Herramienta para generar de forma automática un programa de acceso y mantenimiento a tablas DB2 y su documentación asociada.

- Metalenguajes generadores de COBOL. Generadores de programas COBOL a partir de otros lenguajes con un nivel de abstracción mayor, cómo por ejemplo FAST o

GENESIS. Con estos lenguajes de más alto nivel que el COBOL, facilita la codificación al programador.

***(*) El proyecto se centrará en los dos primeros***

Estas herramientas se pueden desarrollar utilizando paradigmas [5] MDD(Model Driven Development) y MDA(Model Driven Architecture) ya que se pueden establecer diferentes modelos de abstracción para convertir código fuente en información útil, tal como documentación técnica, entornos de prueba, validaciones, etc y [6] otros paradigmas de generación automática de código, tales como el uso de plantillas, código base o técnicas de inyección de código.

En la figura 6 se enumeran las tareas que se realizarán en las distintas fases del desarrollo:

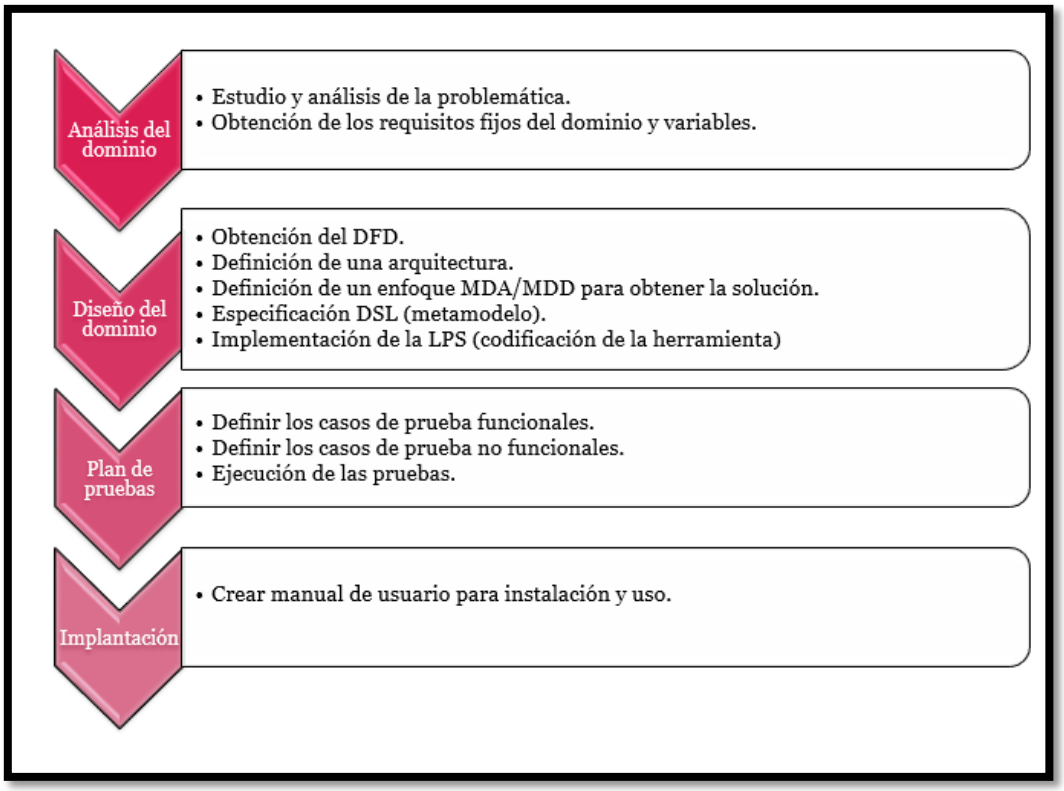



*Figura 6. Tareas propuestas en las distintas fases de desarrollo*

En el análisis y desarrollo de la solución se utilizarán las siguientes técnicas/métodos:

- **MDA**. Se utilizará una arquitectura de desarrollo dirigido a modelos (MDA)
- **DSL**. Se transformará un metalenguaje de alto nivel de abstracción
- **Programación generativa [7]**. El producto final será un modelo generativo capaz de sintetizar todos los programas de una familia a partir de especificaciones de

alto nivel de abstracción mediante un metalenguaje. En esta programación se definen 3 espacios diferenciados:

- **El espacio del problema**. Uno o varios DSLs para facilitar la especificación abstracta de los programas.
- **El espacio de la solución.** Son los marcos de trabajo (interfaces) utilizados para la generación de los distintos objetos finales.
- **El espacio de configuración.** Conversión final a los programas objeto.

- ***ANÁLISIS DE DOMINIO → MODELOS MDA → GENERADORES.*** El reto en el análisis dominio [8] se encuentra en delimitar el ámbito de este dominio de los productos que se deben generar. Esta delimitación se puede dividir en dos grandes grupos:
  - Características fijas (o requisitos). Se deben establecer las partes fijas del producto.
  - Características variables (o requisitos). Identificar las variables que puede soportar el producto final.

Los beneficios esperados son los siguientes:

- Reducción de costes. Mayor inversión inicial pero menor inversión en la generación de los productos, por lo que, con el tiempo de producción, esta inversión se convertirá en beneficio y quedará amortizada.

- Calidad del producto. El producto final puede ir evolucionando y ganando en calidad, gracias a la retroalimentación.

- Menos errores. Los errores detectados, se eliminan y no surgirán para futuras generación.

## *1.4 Arquitectura planteada*

La arquitectura que será planteada para la solución estará basada en un enfoque MDA.

El paradigma MDA (Model Dirven Architecture) es utilizado para la resolución de problemas basados en modelos, utilizando la abstracción para resolver el problema.

Para el caso del estudio se utilizará una representación con un lenguaje de dominio específico (DSL) [6], donde se definirá un modelo de alto nivel textual con una descripción específica de las columnas y la sintaxis que deberá ser utilizada.

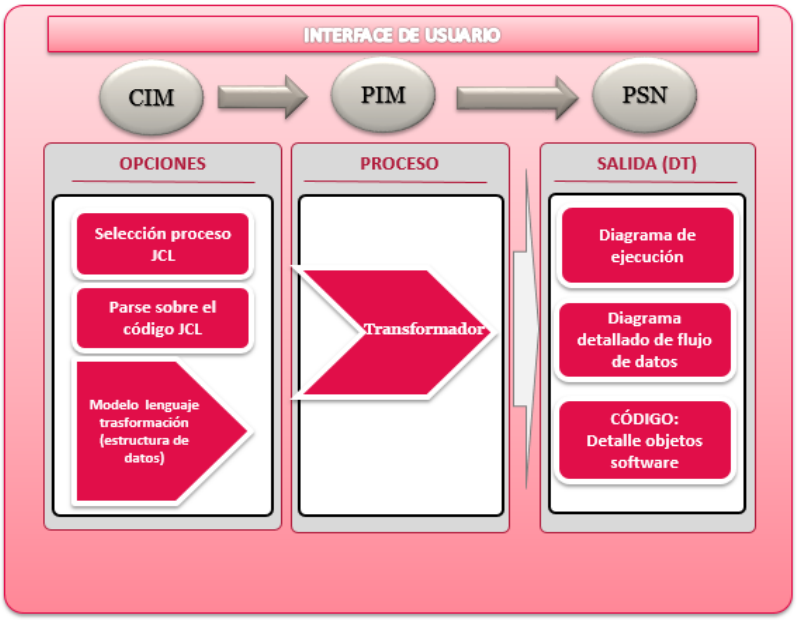
En la figura 7, se puede observar la arquitectura que será utilizada en la solución, utilizando los 3 niveles básicos de MDA:


*Figura 7. Arquitectura en 3 niveles MDA*

En anterior figura, se pueden observar 3 grandes grupos:

- **CIM** (Modelo Independiente de la Computación), formado por la estructura de datos definida en el modelo formal que será transferido y validado por el software desarrollado).
- **PIM** (Modelo Independiente de la plataforma). El lenguaje utilizado para la transformación es VB bajo la plataforma de la hoja de cálculo Excel.
- **PSM** (Modelo específico de la plataforma), modelo textual que representa las estructuras generadas de forma automática, todas ellas unificadas en un documento Word con tipología de diseño técnico.

Las arquitecturas con el primer nivel de abstracción se relacionan en la tabla 1:

| Arquitectura | Descripción |
|---|---|
|  | **Figura 8:**<br><br>OPCIONES. Se seleccionará el JCL objeto de ser transformado a documento.<br>PROCESO. En el proceso se realizará la 'transformación', aplicando el metamodelo que permitirá la transformación de las instrucciones JCL en diagramas, apartados, etc. a una estructura definada.<br>SALIDA. Modelo objetivo final, que será documento técnico. |

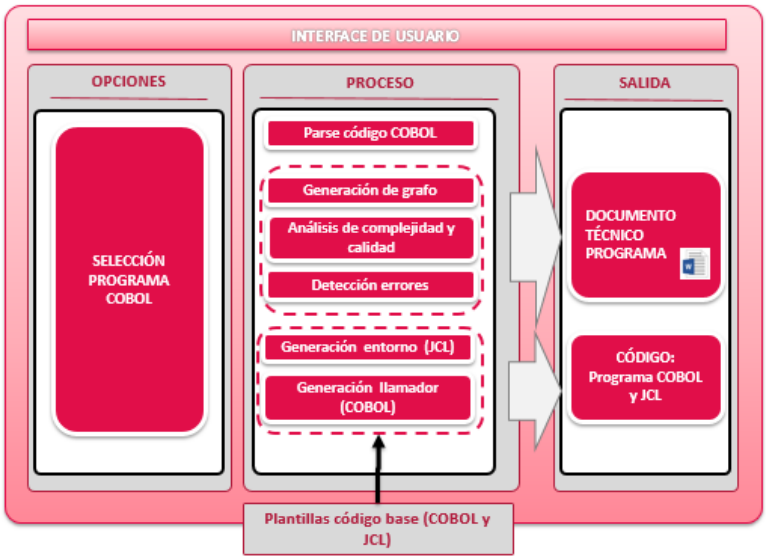
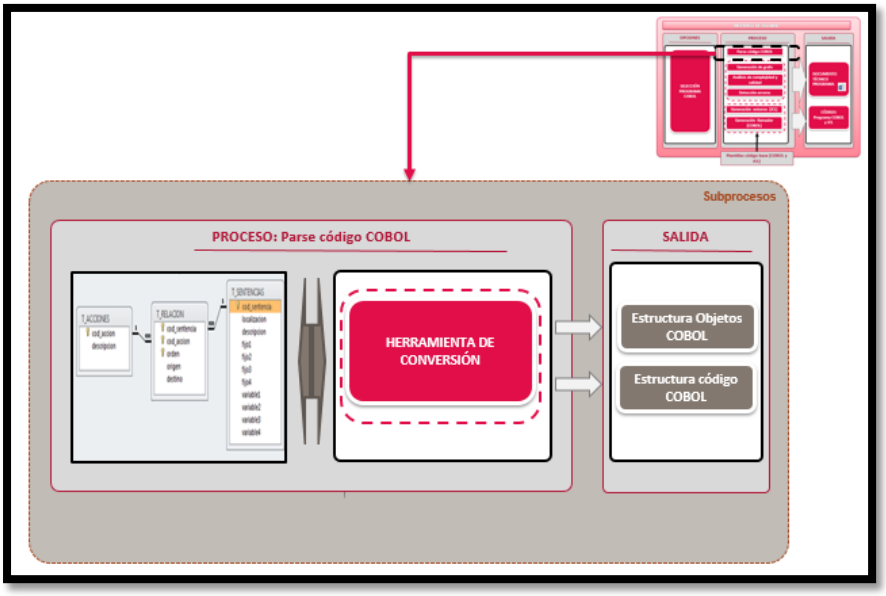
| | |
|---|---|
| *Figura 8. Arquitectura en 3 niveles MDA para GAD* | |
|  <br>*Figura 9. Arquitectura en 3 niveles MDA para el análisis de programas cobol* | **Figura 9:**<br><br>OPCIONES. En la primera opción, se seleccionará el programa objeto de ser analizado.<br><br>PROCESO. En el proceso se realizará la 'transformación', aplicando el metamodelo para cada una de las funcionalidades.<br><br>SALIDA. Modelo objetivo final. |

*Tabla 1. Arquitecturas de primer nivel de abstracción*

Dada la complejidad de la arquitectura correspondiente al análisis de programas COBOL, se realiza una mayor granulación, dando lugar a sub-arquitecturas con abstracción más detalladas dependiendo de la funcionalidad. A continuación, se muestran las más destacadas en la tabla 2:

| Sub-Arquitectura | Descripción |
|---|---|
|  <br>*Figura 10. Sub-arquitectura parse COBOL* | **Figura 10:**<br><br>Apoyándose en unas tablas paramétricas, se descompondrá el código de entrada en elementos más simples, para finalizar con las estructuras de los distintos metamodelos definidos. |

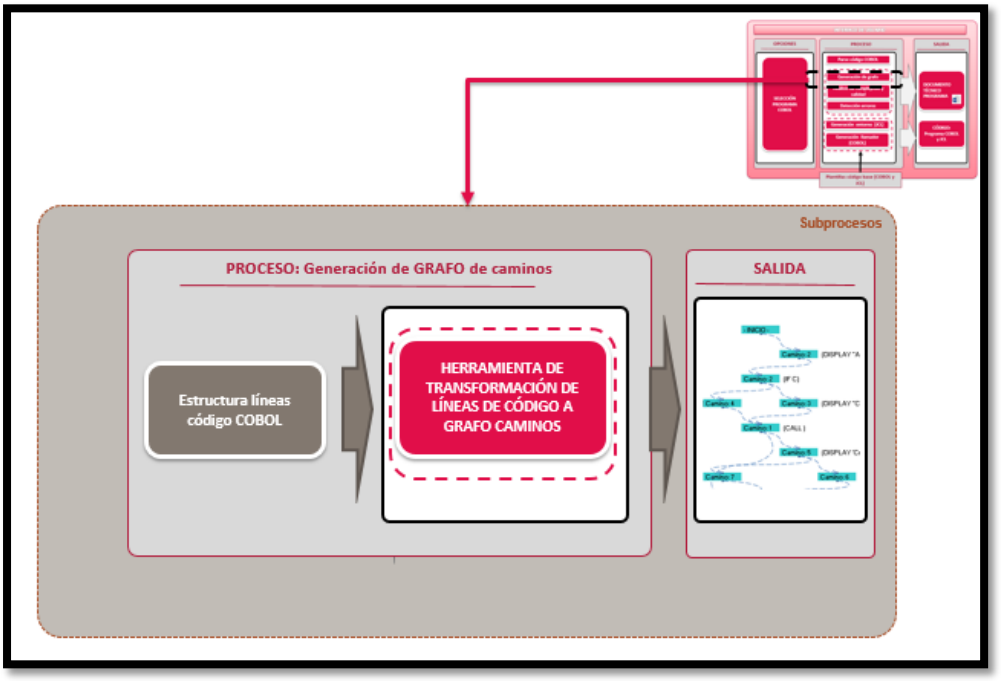



*Figura 11. Sub-arquitectura generación GRAFO*

**Figura 11:**

Partiendo de metamodelos definido en la estructura de código Cobol, se generará un diagrama de caminos.

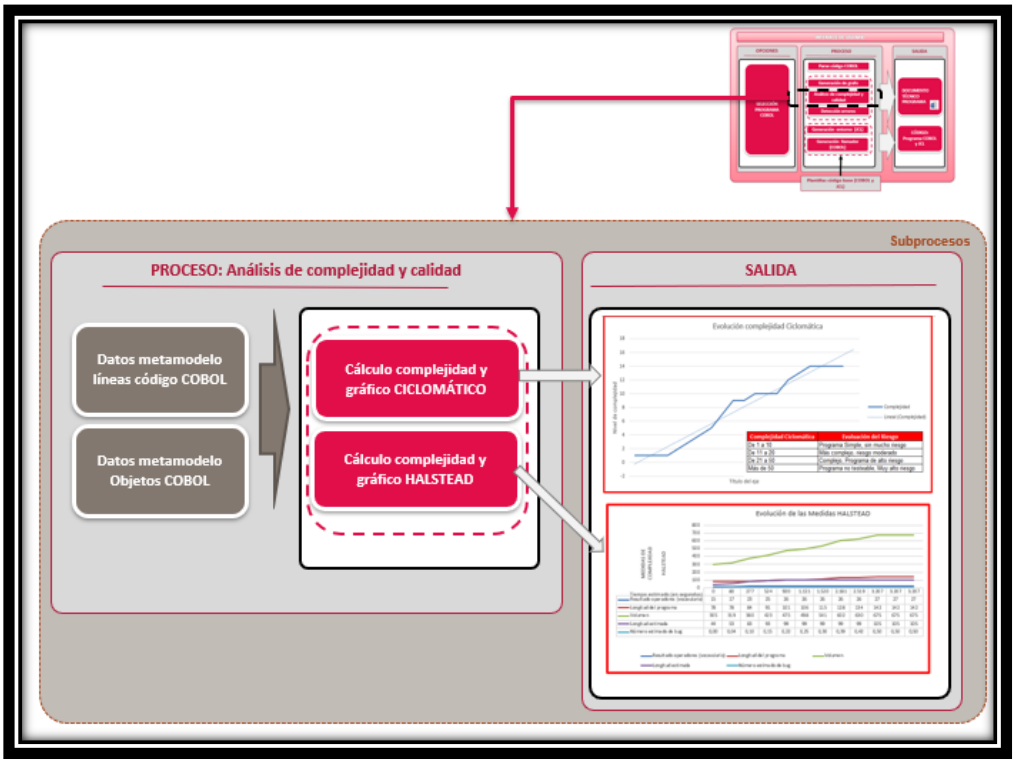



*Figura 12. Sub-arquitectura complejidad y calidad*

**Figura 12:**
Partiendo de las estructuras de los distintos metamodelos, se generarán unas gráficas plasmando las cifras cuantitativas obtenidas al aplicar las diferentes técnicas de análisis de complejidad y calidad. Estás técnicas están basadas en de complejidad ciclomática y en Healdstead.

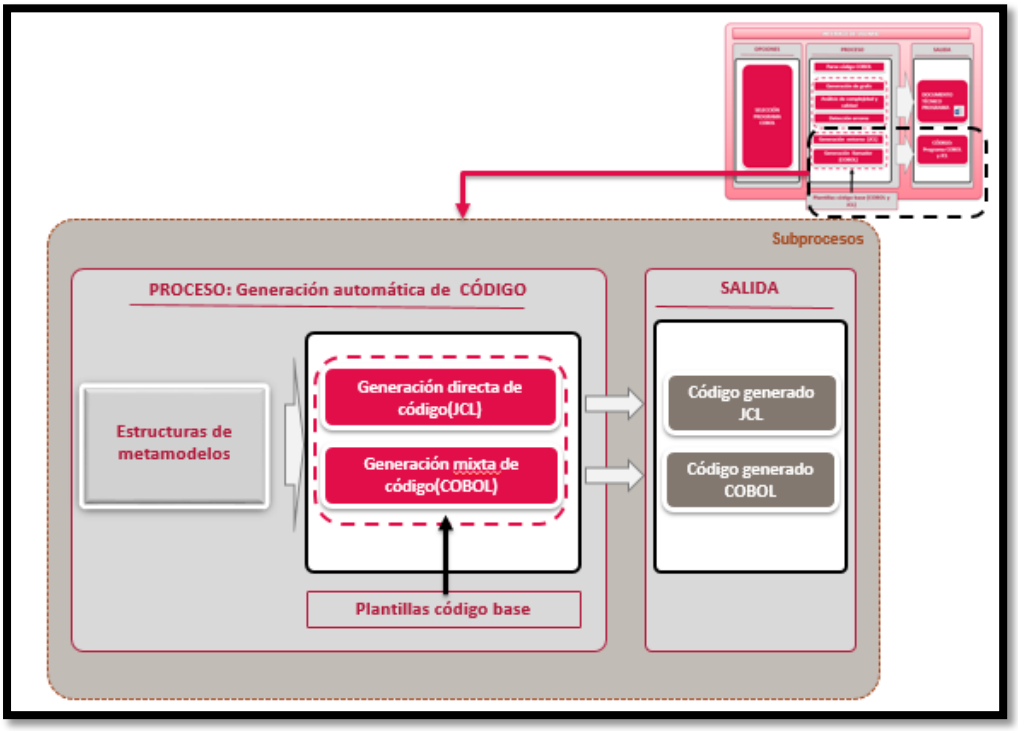



*Figura 13. Sub-arquitectura GAC (COBOL y JCL)*

**Figura 13:**

En esta sub-arquitectura, se obtiene el código de salida necesario para el entorno de pruebas. Para está generación se aplican técnicas de metamodelos, plantillas de código base e inyección de código

*Tabla 2. Sub-arquitecturas de segundo nivel de abstracción*

# CAPÍTULO 2

# 2. Estado del arte

En esta sección se realizará un análisis de los lenguajes utilizados, métricas de calidad aplicadas y las herramientas del mercado disponibles para realizar parte de las funcionalidades descritas en el objetivo del proyecto

En la tabla 3 se enumeran todas las herramientas y lenguajes que serán tratadas en esta sección:

| Lenguaje/ Herramienta | Descripción |
|---|---|
| **COBOL** | Lenguaje de programación estructurada, creado por Grace Hopper en 1959 y utilizado principalmente para Software orientado a negocios y en sistemas Mainframes. |
| **VB** | Lenguaje de programación visual creado por Alan Cooper para Microsoft. Es un lenguaje de programación dirigido por eventos, evolución del BASIC. |
| **JCL** | Lenguaje de control de trabajos batch para comunicarse con sistemas operativos Mainframes. |
| **Genesis** | Herramienta diseñada para la construcción de programas COBOL a través de un metalenguaje. |
| **FAST** | Plataforma basada en eclipse para la generación de programas COBOL a través de plantillas y metalenguaje. |
| **GeneXus** | Generador de aplicaciones B2B y B2C (bussiness to consumer). |
| **CodeDOM** | Generador de código basado una abstracción completa del lenguaje a convertir. |
| **XSLT** | Uso de archivos XSLT para convertir plantillas. |
| **gvNIX** | Herramienta para el desarrollo rápido de aplicaciones. |
| **JUnit** | Framework open source muy estandarizado orientado a realización de pruebas unitarias para aplicaciones desarrolladas en Java. |

| | |
|---|---|
| **NUnit** | Igual que Junit pero orientada al mundo .net.una framework open source de pruebas para Microsoft .net. |
| **Xpediter** | Es una herramienta que permite depurar un programa a través de una depuración ejecutando instrucción a instrucción y permitiendo parametrizar campos. |
| **Abend-AID** | Herramienta para analizar cancelamientos en sistemas Mainframes. |

Tabla 3. Relación de lenguajes y herramientas de GAD y GAP en el mercado

A continuación en la tabla 4, se enumeran las métricas que serán analizadas en el apartado 2.3, centrándose en la medida Healstead y ciclomática, que serán las utilizadas en el desarrollo de la herramienta para establecer la calidad y complejidad del software analizado.

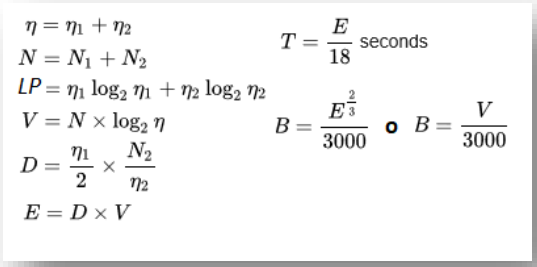
| Tipo de métrica | Descripción |
|---|---|
| Tamaño por número de líneas | La complejidad se establece por el número de líneas de código que tiene el programa.<br>Existen variantes, en las cuales se determina código útil para tener en cuenta la medida.<br>No se considera muy fiable. |
| Medida Healstead | Introducidas por Maurice Howard Halstead en 1977. Basadas también en parte por el tamaño del software, se toman otrs parámetros del programa para asignar la complejidad y calidad del software analizado.<br>  <br>$\eta = \eta_1 + \eta_2$<br>$N = N_1 + N_2$<br>$LP = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$<br>$V = N \times \log_2 \eta$<br>$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$<br>$E = D \times V$<br>$T = \frac{E}{18}$ seconds<br>$B = \frac{E^{\frac{2}{3}}}{3000}$ o $B = \frac{V}{3000}$<br>(*) desarrollado en apartado 2.3.1 |
| Medida Q de Chapin | Basadas en la estructura y flujo de datos, asignando diferentes tipologías a los datos para realizar los cálculos. |
| Ciclomática | Basadas en el flujo de control, determina el número de caminos de ejecución que tiene un software, estableciendo 4 niveles de complejidad. |

Tabla 4. Medidas de calidad y complejidad analizadas

## *2.1 Lenguajes utilizados en el proyecto*

### 2.1.1 COBOL

COBOL es el acrónimo de COmmon Business-Oriented Language, creado por el Short Range Committee en 1959 altamente influenciado por Grace Hopper.

El objetivo principal del COBOL, era el de ser un lenguaje de programación universal que pudiera ser usado en cualquier ordenador y que estuviera orientado a los negocios.

En la actualidad existen varios compiladores tanto para Windows [9] como para Linux, entre los cuales podemos encontrar los siguientes:

- **MicroFocus NetExpress.** IDE ya moderno permitiendo interactuar con Java, EJB, C. También OO COBOL (orientado a objetos)).
- **MicroFocus Visual COBOL.** Para Visual Studio [10] y Eclipse (el IDE actual, con WebServices).
- **Fujitsu COBOL.** NetCOBOL for Windows, NetCOBOL for .NET, PowerCOBOL (forma parte del paquete NetCOBOL for Windows, creando aplicaciones GUI basadas en controles ActiveX, soporta WinAPI).
- **GNU Cobol.** Antiguo Open COBOL, que es Open Source.
- **Raincode COBOL**. Clásico compilador.

Gracias a la excelente gestión de archivos y de tipo de datos, el COBOL tuvo una gran expansión que se mantuvo hasta mediados de los 80 que fue cuando comenzó a quedarse anticuado con respecto a nuevos paradigmas de programación.

Pese a esta desactualización, lejos de quedar en desuso, el COBOL sigue siendo uno de los pilares principales de grandes organizaciones, cómo por ejemplo las entidades bancarias, ya que, a parte de la gran capacidad de procesamiento de información, también mantiene un nivel de seguridad muy elevada al estar ubicados en los grandes sistema Mainframes.

Este lenguaje siempre ha estado muy relacionado con el software de grandes compañías, gracias en un principio por su gestión de grandes volúmenes de información y por ser uno de los pocos lenguajes soportados en entornos mainframes.

Aunque con la evolución de nuevas tecnologías y nuevas arquitecturas, su utilización ha ido disminuyendo a lo largo de los años y en la actualidad es utilizado solamente para entornos centralizados UNIX y MVS y en las capas más internas de acceso a datos o para entornos Batch.

La evolución de las arquitecturas en estos grandes sistemas ha tenido mucho impacto en la disminución de la utilización del cobol. Podemos diferencias 3 grandes etapas en esta evolución, tal como se muestra en la figura 14:

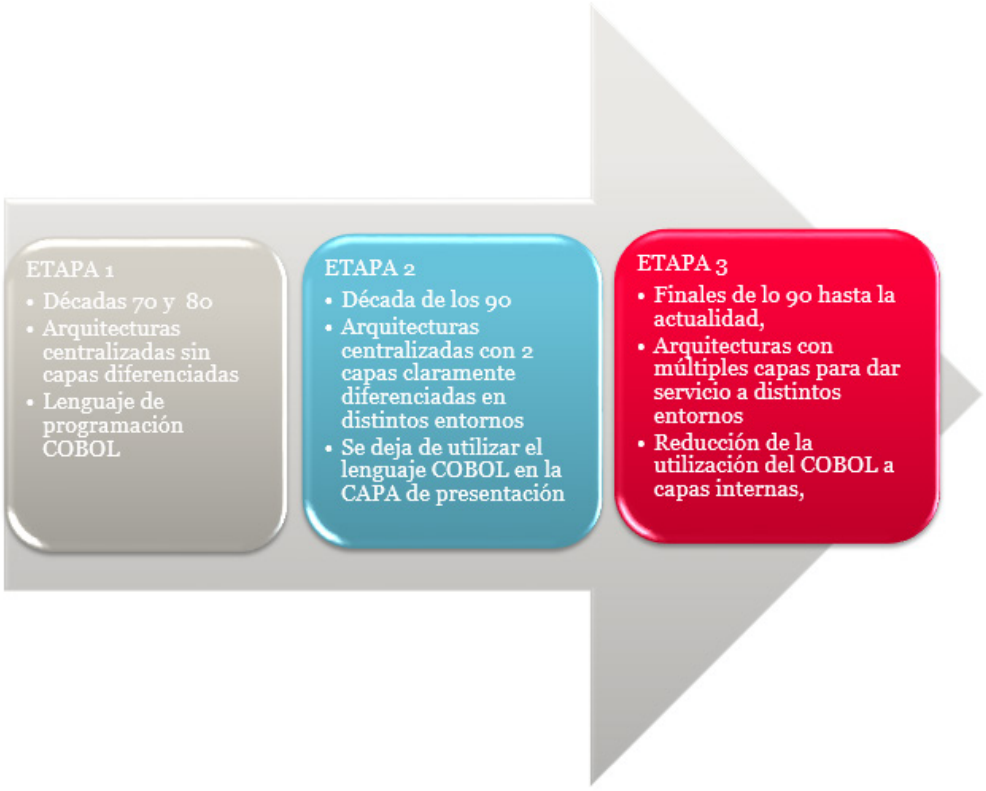



*Figura 14. Etapas destacadas de la evolución del lenguaje COBOL*

### 2.1.2 Lenguaje VB

El VB es el lenguaje elegido para realizar la codificación. El lenguaje Visual Basic [11] fue creado por Alan Cooper para Microsoft. Es un lenguaje de programación dirigido por eventos, evolución del BASIC [12] con importantes mejoras sobre éste y que apareció para ofrecer una alternativa sobre lenguajes más complejos, como por ejemplo el C/C++.

A partir de la versión 6.0, VB se integró en la plataforma .NET, donde perdió la identidad de lenguaje único para pasar a integrar un paquete de productos dentro de Microsoft .NET (Visual Basic .NET). En la tabla 5 se describen estas versiones.

| Versión | Descripción y mejoras de la versión |
|---|---|
| Versión 1.0 | Visual Basic 1.0 para MS-DOS fue liberada en septiembre de 1992. Poco popular, este lenguaje no era compatible con Visual Basic para Windows.<br>Esta versión es la menos conocida y menos utilizada. |
| **Versión 2.0** | |

| | |
|---|---|
| **Versión 3.0** | Esta versión apareció en el mercado en el verano del 93, con dos versiones, una para profesionales y otra estándar el resto del público.<br>En esta versión se incluyó Microsoft Jet Database Engine 1.1, permitiendo accesos a bases de datos Access. |
| **Versión 4.0** | Publicada en agosto de 1995. Se incluye IDE modo interprete para poder ir probando el programa según se desarrolla.<br>Otra de las mejoras más relevantes, es que se crearon dos versiones, una versión para 16 bits y otra para 32 bits.<br>Inclusión de controles denominados VBX por los nuevos OCX |
| **Versión 5.0** | En febrero de 1997, Microsoft lanzó Visual Basic 5.0, versión que generaba programas de 32 bits exclusivamente.<br>Integración de diseños de formularios de Windows. Permite usar con facilidad la plataforma de los sistemas Windows, dado que tiene acceso prácticamente total a la API de Windows, incluidas librerías actuales.<br>Incorporación de características de los lenguajes orientados a objeto (excepto herencia) |
| **Versión 6.0** | Visual Basic 6.0, publicado a mediados de 1998, muy mejorado, incrementó el número de áreas2 e incluyó la posibilidad de crear aplicaciones basadas en Web.<br>Esta versión genera código ejecutable directo en 32 bits. |
| **.NET** | Incorporación al paquete de Microsoft .NET. |

*Tabla 5.* Versiones más destacasas del lenguaje VB

### 2.1.3 Lenguaje JCL

Job Control Language [13] o lenguaje de control de trabajo, es el encargado de definir el entorno de ejecución de los programas incluidos en una cadena.

El lenguaje de control de trabajos (JCL) se utiliza para comunicarse con él sistema operativo, en este caso con el sistema os/390-IBM de Mainframes.

El sistema operativo OS/390 consiste en un programa de control de bases (BCP) con un subsistema de entrada/salida de trabajos (JES2 o JES3). Para que el sistema operativo

procese un programa, el programador debe realizar ciertas tareas de control de trabajos. Estas tareas se realizan a través del lenguaje de control de trabajos (JCL). Las tareas:

Estas tareas son:

- Entrada de trabajos.
- Procesar trabajos.
- Petición de recursos.

Estructura principal de JCL.

En la figura 15 se muestra la estructura del código JCL con sus principales bloques de información:

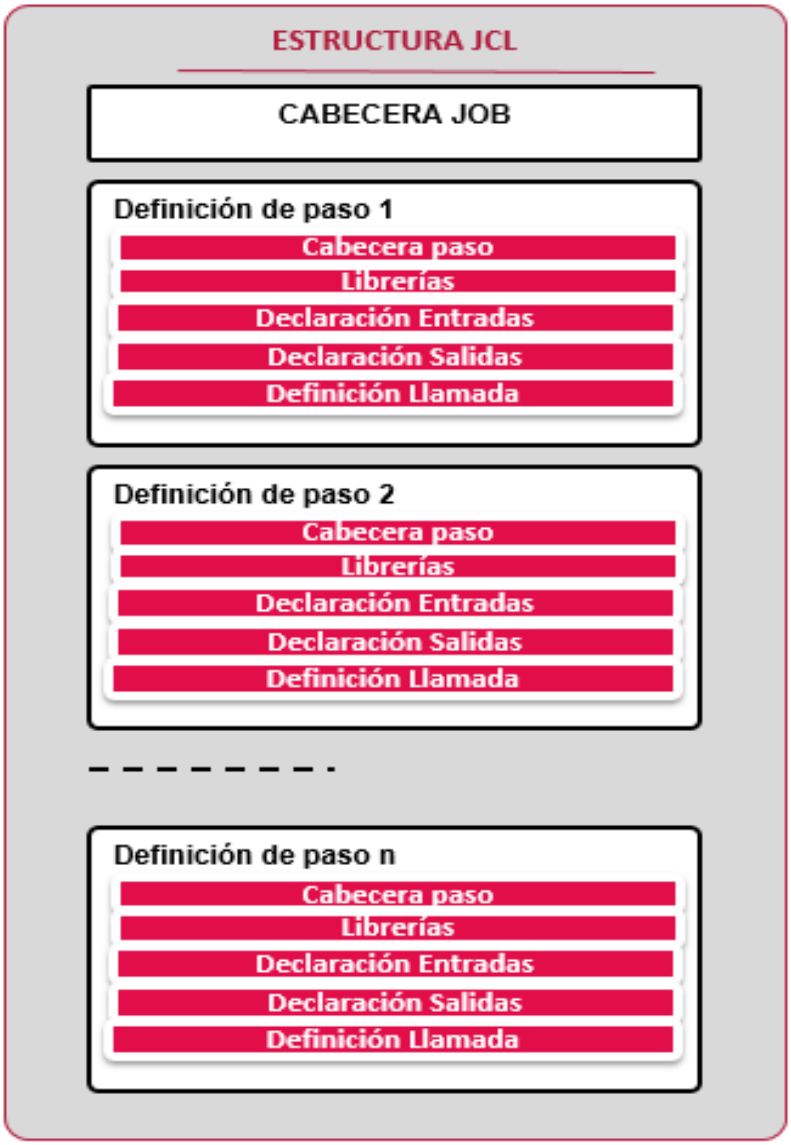



*Figura 15. Estructura principal de un proceso batch escrito en JCL*

Sentencias más utilizadas en JCL (descritas en tabla 6)

| Sentencia | Descripción | Ubicación |
|---|---|---|
| Comentario | Incluir comentarios dentro de la cadena de Jcl. En las posiciones 1, 2 y 3 debe informarse de //* y a continuación el comentario que se desee exponer. | Todas |
| Sentencia JOB | Es el indicador del inicio de JOB. La sentencia JOB consiste en los caracteres // situados en las columnas 1 y 2, cuatro campos: nombre, operación (JOB), parámetros y comentarios.<br><br>Se requiere una sentencia JOB por cada trabajo. | Cabecera |
| Sentencia PROC | Indica los distintos procedimientos que tiene la cadena. Estos procesos se corresponden a programas predefinidos por el SO como por ejemplo COPIAS, SORT, DESCARGAS, ETC o con programas definidos por los desarrolladores. Un JOB puede tener n procedimientos. | Cabecera |
| Sentencia EXEC | Identificación del programa que se va a ejecutar. | Cabecera |
| Sentencia DD | Definición de datos. Utilizada para identificar y configurar las interfases de E/S a los programas. | Pasos |
| DD especiales | Definición de datos que son especiales | Todas |
| Delimitadoras | Utilizada para delimitar el final de los datos o registros. | Todas |
| CLASS | Indica la clase de ejecución del JOB. Las clases de ejecución estándar son:<br>• 'D'. Ejecuciones en desarrollo sin acceso a BBDD.<br>• 'U'. Ejecuciones en desarrollo con acceso a BBDD.<br>• 'O'. Ejecuciones en producción con BBDD.<br>• 'C'. Ejecuciones en producción sin BBDD. | Cabecera |
| MSGCLASS | Se utiliza para especificar la clase de salida para visualización de los mensajes del JOB. (Impresión o por consola) | Cabecera |

| | | |
|---|---|---|
| MSGLEVEL | Control y mensajes relacionados con la salida del trabajo. | Cabecera |
| NOTIFY | Usuario al que se debe notificar la salida del JOB | Cabecera |
| REGION | Asignación de memoria virtual para la ejecución del JOB | Cabecera |
| PGM | Nombre del programa que va a ser ejecutado en el paso. | Pasos |
| PARM | Uso para enviar parámetros al área de intercambio de datos del programa | Pasos |
| AVGREC | Especifica las unidades de asignación para los ficheros de salida, para indicar unidades, miles o millones. | Pasos |
| BLKSIZE | Especificación de la longitud máxima por bloque | Pasos |
| DISP | El parámetro DISP se usa para describir el estado de un conjunto de datos e indicar el estado en el cual se quieren dejar una vez finalizado el JOB. | Pasos |
| DSNAME | Nombre externo al que se hace referencia para los datos de E/S | Pasos |
| LRECL | Definición de longitud para los registros | Pasos |
| RECFM | El parámetro RECFM se usa para especificar el formato y las características de los registros en un nuevo conjunto de datos | Pasos |
| SPACE | El parámetro SPACE se usa para solicitar espacio para un nuevo conjunto de datos en un volumen de acceso directo | Pasos |
| SYSOUT | El parámetro SYSOUT se usa para identificar un conjunto de datos como un conjunto de datos de salida del sistema. Asigna el conjunto de datos de salida del sistema a una clase de salida. | Pasos |
| UNIT | El parámetro UNIT se usa para informar al sistema el soporte dónde tiene que guardar un conjunto de datos en. Dispositivos permitidos: ROBOT, TAPE, DISCO, CARTUCHO, ETC | Pasos |

*Tabla 6.* Sentencias más destacadas código JCL

## 2.2 Herramientas generadoras de código y pruebas

En el mercado existen multitud de herramientas generadoras de código y pruebas, pero la mayoría de ellas están centradas para JAVA , C, C#, etc.

El abanico ya no es tan amplio para el caso del código COBOL.

En la figura 16 se enumeran algunas de las herramientas generadoras de código que existen en el mercado:

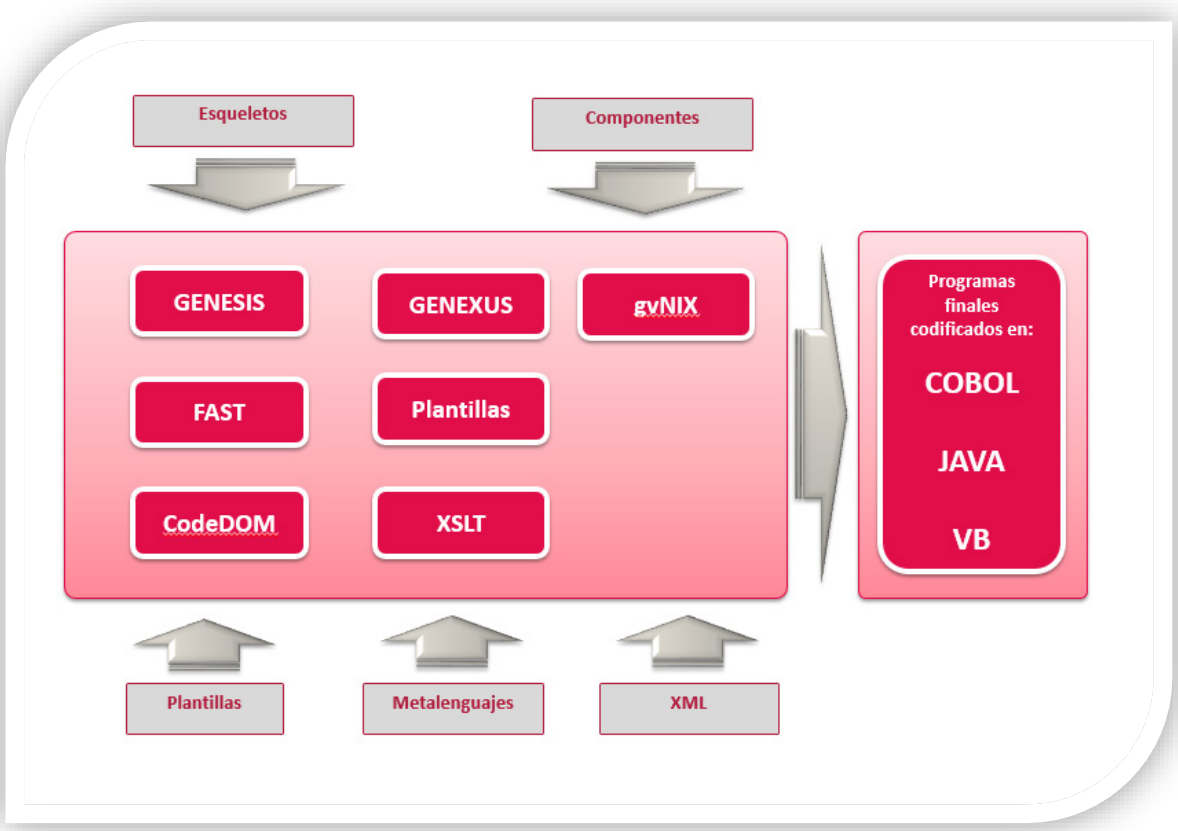



*Figura 16. Herramientas generadoras de código*

### 2.2.1 Genesis.

Es un sistema de fabricación de programas basado en la utilización de componentes y rúbricas, que aprovecha la estandarización y la reutilización de código, para con construir programas COBOL entre los que se encuentran la generación de progRamas de acceso a tablas DB2.

GNSIS fue escrito originalmente sobre Visual Studio 2005 para el Microsoft .Net Framework 2.0. Utiliza tecnología ClickOnce para su instalación y distribución en todos los puestos de la parte cliente que tengan sistema operativo Windows, Framework 2.0

instalado y acceso a la red y a los servidores corporativos donde está instalada la parte servidor.

Los componentes pueden ser diseñados o adaptados por cada organización, para adaptarlos a su arquitectura. Los programadores utilizarán estos componentes ya adaptados para generar sus programas.

El programa GNSIS es un lenguaje de codificación que cuenta con una estructura más simple que un programa COBOL, por lo que teóricamente es más sencillo de mantener que un programa escrito directamente en COBOL.

Los programas genesis se elaboran en el editor de GNSIS, el cual está diseñado especialmente para la construcción de los programas. Para crear un progama se debe seleccionar un esqueleto (componente principal de GNSIS) y al esqueleto se le van incorporando componentes para ir dando forma al programa. En la tabla 7 se enumeran los elementos principales del GNSIS.

| Elementos y características principales | Descripción |
|---|---|
| **Editor de GNSIS** | Se trata de una aplicación diseñada para la construcción de los programas en GNSIS, en la cual se incluye componentes de edición, impresión, generación, etc. (es el entorno de trabajo). |
| **Esqueleto** | Componente principal de GNSIS sobre el cual se construirá la base del programa. Dependiendo del esqueleto escribiremos una tipología de programa (repetitivo, enfrentamiento, acceso a BD, etc). |
| **Componente** | Se denomina a las macros que se utilizan dentro de los esqueletos. |
| **Lenguaje generado** | COBOL |

Tabla 7. *Elementos principales GNSIS*

La herramienta se debe instalar directamente en los equipos (no entornos distribuidos), utilizando una plataforma específica. Mediante la selección de la plantilla correspondiente y la inclusión de parámetros obligatorios mediante un lenguaje específico definido en GENESIS se genera un código fuente.

Los principales inconvenientes se describen en la tabla 8:

| Tipo problema/inconveniente | Descripción |
|---|---|
| **No dinámica** | La herramienta no es dinámica y no realiza una validación del programa COBOL generado por lo que éste puede contener errores de prevalidación |
| **Ganancia de tiempo limitada** | La ganancia en tiempo de producción (creación de los programas) es muy pequeña con respecto a la reutilización directa de programas COBOL (no pasa del 10%) |
| **No libre de errores** | La herramienta no asegura que programas generados funcionen de forma correcta ya que los parámetros de inclusión es muy extensa |
| **Necesidad de formación** | Para poder utilizar la herramienta es necesaria una formación previa para su utilización de unas 50-70 horas para que le programador pueda comenzar su utilización. |
| **Coste** | Es una herramienta de alto coste, que necesita licencias, por lo que parte del ahorro en la automatización se irá a la compra y mantenimiento de la herramienta. |

*Tabla 8. Análisis de inconvenientes GNSIS*

Cómo puntos positivos, nos encontramos que todos los lenguajes generados contienen la misma estructura, por lo que su mantenimiento resulta más sencillo.

### 2.2.2 FAST

Es una plataforma está basada en eclipse, con uso en distribuido. A través de metalenguaje específico llamado FAST, con la utilización de ventanas de ayuda y de plantillas en forma de esqueletos, se generan programas en diferentes tipologías COBOL (repetitivos, enfrentamientos o acceso a BD).

La plataforma es dinámica y una vez generado el programa COBOL permite realizar su validación, lo que permite ahorrar MIT en entorno mainframes que suelen ser más costosos.

Esta herramienta busca la reducción de tiempo en la codificación de programas COBOL y facilitar el mantenimiento de éstos.

Otras características de Fast es que tiene Integración con Topaz Workbench, herramienta de Compuware que provee un interfaz en eclipse para sus herramientas (File-Aid, Xpediter, Code coverage), y con los editores de código cobol e incorporación de JCL (Job Control Language).

**ESQUELETO + COMPONENTE = PROGRAMA COBOL**

Genera una estructura COBOL fácil de mantener, pero al igual que en GENESIS no asegura que el programa que generemos funcione correctamente.

Características de FAST a continuación, en la tabla 9.

| Elementos y características principales | Descripción |
|---|---|
| **Plataforma** | Uso de la plataforma eclipse |
| **Lenguaje generado** | COBOL |
| **Revisión de errores** | Previsualización de errores gramaticales en la inclusión de componentes |
| **Disponibilidad** | Disponibilidad de la estructura de esqueleto utilizado y sus componentes |
| **Entorno dinámico** | Cuenta con un entorno dinámico para la creación de programas COBOL, permitiendo realizar réplicas, visualizaciones, etc. |
| **Herramientas potentes** | Al usar eclipse, cuenta con potentes herramientas de búsqueda y reemplazo de cadenas, Posibilidad de editar un elemento en varias pestañas |
| **Plugins** | Permite la utilización de plugins de eclipse (Endevor, Xpediter, Fail Aid, DevEnterprise, etc). |
| **Programación visual** | Permite realizar una programación mixta entre visual y por entrada de línea. |
| **Macros** | Permite la definición de macros. |
| **Compilador** | Inclusión de compilador COBOL para validar el programa generado en COBOL, lo que ahorra compilaciones extras en los sistemas Mainframes |

*Tabla 9. Elementos y características de FAST*

Los principales inconvenientes que ven en Fast se reflejan en la tabla 10:

| Tipo problema/inconveniente | Descripción |
|---|---|
| **Autocorrección** | No permite la desactivación de la autocorrección, lo que pueda causar problemas a usuarios no habituados a ella |
| **Editor complejo** | El editor tiene muchas ventanas y muchas opciones, lo que resulta complejo |
| **Lenguaje completo** | Es necesario un curso de formación de unas 50-70 horas. |
| **Dificultad problemas complejos** | Si la funcionalidad del programa COBOL es compleja o no se adapta a los estándares, es muy laborioso su codificación. |
| **Coste** | Es una herramienta de alto coste, que necesita licencias, por lo que parte del ahorro en la automatización se irá a la compra y mantenimiento de la herramienta. |

Tabla 10. Análisis de problemas del FAST

### 2.2.3 GeneXus

Es un generador multilenguaje [14][15] para el desarrollo de aplicaciones B2B (business to business). También permite la generación de Web Panels. Lo que permite crear aplicaciones B2C (bussiness to consumer).

Para el desarrollo de aplicaciones de tipo bussiness to computer cuenta con el objeto Web Panel para generadores en C/SQL, Visual Basic y Java, pero en java se utiliza tecnología Servlets y en C y VB se utiliza CGI-BIN..

Características de Genexus en la tabla 11:

| Características | Descripción |
|---|---|
| **Lenguaje generado** | VB, JAVA, C#, COBOL, RPG, Ruby |
| **Plataforma de ejecución** | GeneXus soporta su ejecución en las siguientes plataformas: Java/J2EEE, .NET Compact Frfamework, Android, IOS, BlackBerry |
| **Publicación** | La primera versión del generador Java se publicó en septiembre de 1999. |
| **Diversidad de objetos** | Permite la generación de todo tipo de objetos, excepto objetos de tipo menú. |
| **Versión Ceibo beta 3** | Esta versión es capaz de soportar transacciones de varios niveles. |
| **Reglas** | Permite la ejecución de reglas en las transacciones. |
| **GXDB++** | Apoyo en GXDB++ (clases java de acceso a BD) para realizar acceso a base de datos. Las GXBD++ deben estar sincronizadas con el modelo. |
| **Uso de máquinas** | A la fecha, las aplicaciones generadas con el generador |

| | |
|---|---|
| **virtuales** | Java funcionan correctamente utilizando el JDK de Sun Microsystems, la máquina virtual del Internet Explorer 4 y el Netscape Navigator versión 4.06 o posterior. Es posible también usar la máquina virtual del IE4 con el IE3. |
| **Simplicidad** | Según sus creadores, el desarrollo resulta más simple, efectivo y productivo. |
| **Obsolescencia** | Aplican seguro contra la obsolescencia tecnológica. |
| **Menor mantenimiento** | Menor coste y tiempo de mantenimiento. |
| **Equipo** | Potencia el trabajo en equipo |
| **Productivo** | Al disponer de generación automática del 100% la productividad aumenta. |
| **Adaptabilidad** | Gracias a su rapidez y agilidad, permite una gran adaptabilidad a los cambios de negocio y a la evolución tecnológica. |

*Tabla 11. Características de GenXus*

En la tabla 12 se enumeran los principales problemas asociados a Genxus:

| Tipo problema | Descripción |
|---|---|
| **Solamente genera Aplicaciones cliente-servidor** | El generador solo generará aplicaciones cliente-servidor (no habrá una versión que genere programas con acceso a base de datos local como Access o DBFs). Las aplicaciones pueden generarse en 2 capas utilizando JDBC (el equivalente Java a ODBC) o en 3 capas utilizando CORBA, DCOM o RMI como protocolo de comunicación entre las capas. |
| **Nombres muy largos** | El generador Java genera programas fuente con nombres largos (de más de 8 letras), por lo que el modelo debe crearse en una unidad de disco que soporte nombres largos. |
| **Problemas instalación** | Problema de instalación si en el equipo existen versiones anteriores de GeneXus. |
| **Necesario curso de formación** | Es necesario curso de formación para que el programador pueda empezar a generar programas con la herramienta. |
| **Coste** | Es una herramienta de alto coste, que necesita licencias, por lo que parte del ahorro en la automatización se irá a la compra y mantenimiento de la herramienta. |
| **Ingeniería inversa** | Problemas [16] con la conversión de caracteres locales y tildes. |
| **Versiones** | No cuenta con una version versión para Linux. |

*Tabla 12. Problemas e inconvenientes de GeneXus*

### 2.2.4 CodeDOM

CodeDOM [17] (Code Document Object Model) es un generador de código que utiliza una abstracción completa de los lenguajes de programación a los cuales va a convertir el código.

El código se escribe a través de las abstracciones definidas en un provider de .NET para un lenguaje VB o C# u otros ***provider*** soportados .

El funcionamiento de CodeDOM se puede ver en la figura 17:

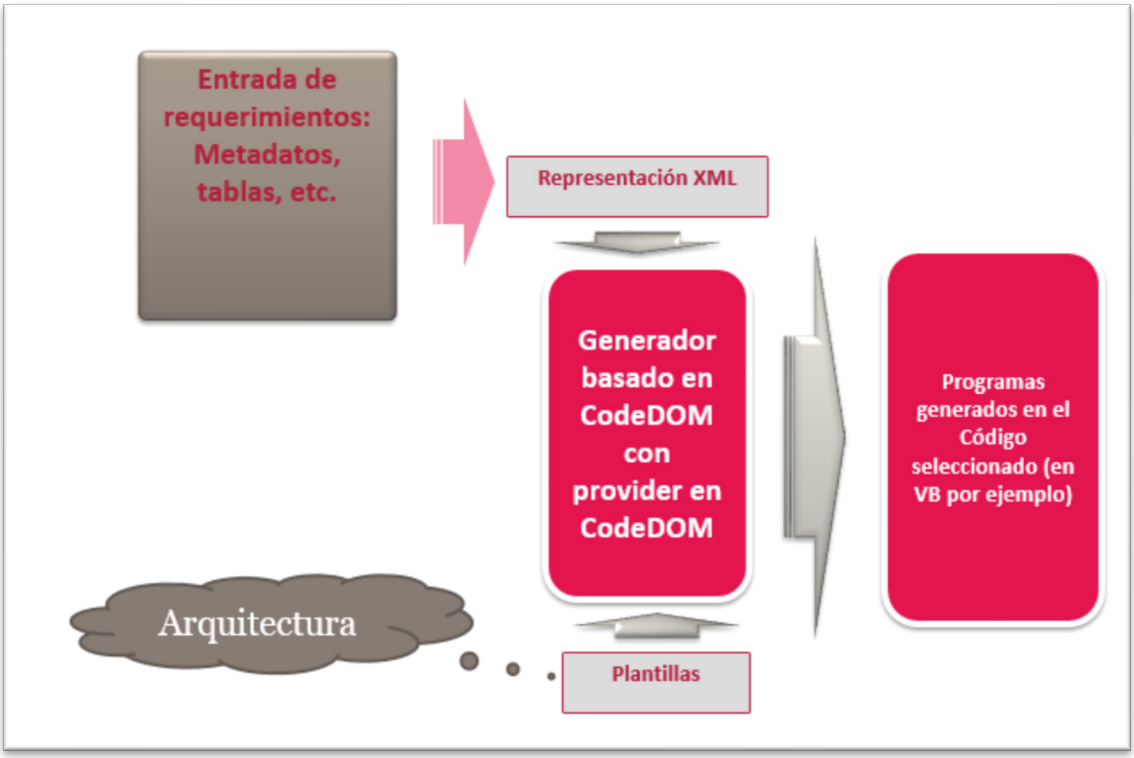



*Figura 17. Funcionamiento de CodeDOM*

La pieza más importante del desarrollo de los generadores es la arquitectura, ya que será la encargada de organizar todas las piezas y pasos necesarios para la generación del código.

Una vez generado un diseño se crearán las plantillas del código las cuales serán utilizadas para producir los programas en cadena, pudiendo usar plantilla XML intermedias que facilitarán esta generación.

Las características principales de CodeDOM a continuación, en la tabla 13:

| Características | Descripción |
|---|---|
| **Basados en arquitecturas.** | Con CodeDOM podemos crear generadores de clase en base a una Arquitectura |
| **Múltiples lenguajes** | Su diseño permite la generación de múltiples lenguajes, |

| | |
|---|---|
| | dependiendo de los provide. |
| **Tiempos** | Su automatización permite reducir los tiempos de producción de software. |

*Tabla 13. Principales características de CodeDOM*

### 2.2.5 Usando XSLT

En este método se usa un archivo de tipo XSLT que será usado como entrada para la conversión de la plantilla del código. Se muestra el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text" encoding="UTF-8" indent="yes"/> <xsl:preserve-space elements="*" /> <xsl:param name="TableName"/> <xsl:template match="/DataSet"> <xsl:apply-templates select="Table[@Name=$TableName]"/> </xsl:template> <xsl:template match="Table"> using System; public class <xsl:value-of select="$TableName"/> { #region Class level declarations<xsl:for-each select="Column"> private <xsl:value-of select="@Type"/> m_<xsl:value-of select="@Name"/>; </xsl:for-each> #endregion #region Public Methods and Properties <xsl:for-each select="Column"> public <xsl:value-of select="@Type"/><xsl:text> </xsl:text><xsl:value-of select="@Name"/> { get { Return m_<xsl:value-of select="@Name"/>; } set { m_<xsl:value-of select="@Name"/> = value; } } </xsl:for-each> #endregion } </xsl:template> </xsl:stylesheet>
```

### 2.2.6 gvNIX

Herramienta de código abierto para el desarrollo rápido de aplicaciones [18].

Las principales características de gvNIX [19] se reflejan en al tabla 14:

| Características | Descripción |
|---|---|
| **Lenguajes que genera** | JAVA |
| **Distribución** | Sprint Roo |
| **Productividad** | Según sus autores, su principal ventaja es el aumento de la productividad en el desarrollo. |
| **Integración de frameworks** | IQuery, Bootstrap 3, Leaflet, Datatables, Dandelion, entre otras. |
| **Multiplataforma** | Se puede instalar en las principales plataformas: Windows, Mac, o Linux |
| **Entorno de trabajo** | Se puede elegir entre STS o Eclipse |
| **Archivos ITD** | Generación no intrusiva de código |
| **Sin dependencias** | No tiene dependencias de entornos de ejecución |
| **Interprete** | Funciona como intérprete de comandos interactivo. |
| **Otras funciones** | Permite exportar/importar servicios WEB, generar informes, ingeniería inversa en base de datos, control de |

| | |
|---|---|
| | concurrencia, auditoría en cambios en BD, histórico de cambios, etc |
| **Arquitectura con avanzada Tecnología** | Uso de tecnología avanzada: Java 6+, Maven, Sprint, Iquery (incluir gráfica con su arquitectura) |

*Tabla 14. Principales problemas gvNIX*

En la tabla 15 se enumeran los principales problemas asociados a gvNIX:

| Tipo problema | Descripción |
|---|---|
| **Requerimientos** | Es necesaria java 6 o superior, SDK y Apache Maven 3 para su correcto funcionamiento. |
| **Complejidad** | Al disponer de multitud de opciones, su uso es complejo. |
| **Formación** | El programador debe formarse antes de su uso y para permitir obtener la productividad que ofertan. |
| **Precio de licencia** | Lleva asociado coste. |

*Tabla 15. Principales problemas gvNIX*

### 2.2.7 Plantillas de generación de código y otros generadores

Existen generadores a base de plantillas de código para modelos de Java simple denominados POJO (Plain Old Java Objects) [20] y para servicios web, aunque estas plantillas no contienen el código definitivo y deben ser adaptadas según características del usuario final.

En la tabla 16 se enumeran otro tipo de generadores de código:

| Generador de código | Usuarios de destino | Finalidad |
|---|---|---|
| **Generador POJO** | Para expertos | Son plantillas, por lo que debe se debe personalizar. |
| **Generador de servicios web (compatible con JDK 6)** | No expertos. Permite crear reglas para webservice. | Para servicios web, arquitecturas SOA (Arquitecturas Orientadas a Servicio) o procesos empresariales (BPM) que utilizan un modelo de datos centrado en XML. |
| **Generador de servicios web JAX-WS 2.1.1** | Basados en estándares. | JAX-WS 2.1.1 RI se proporciona en la distribución y puede desplegarse en Tomcat 7.0. Los tipos Java se serializan y deserializan a teavés de las API JAX-B estándar. |
| **Generador SCA (Service Component Architecture)** | Usuarios específicos. | Generación de archivado SCA basado en un proyecto de aplicaciones de reglas y exportarlo a Rational Application Developer. |

*Tabla 16. Otros generadores*

## *2.3 Medidas de complejidad del software*

La medición del software es el proceso por el cual nos ayuda valorar la calidad del software y la calidad del trabajo realizado por los desarrolladores.

Las métricas son instrumentos que se utilizan para poder medir y valorar la complejidad del Software, pero para qué medir la complejidad.

Es importante disponer de medidas cuantitativas para poder realizar comparaciones entre distintos elementos de software, y mediante la observación y la comparación, se podrá determinar la posibilidad en porcentaje de que el software pueda llegar a fallar, la disponibilidad de adaptabilidad, etc.

Los elementos objeto de realizar la medición, corresponden con Productos software (código o documentación), con el proceso del desarrollo software o medidas para establecer el dominio del problema. En nuestro caso nos centraremos en el estudio de medición de los productos software.

Las métricas se pueden clasificar:

- Métricas de tamaño.
- Métricas de estructura y flujo de datos.
- Métricas de flujo de control.
- Métricas mixtas. Intentan medir simultáneamente varias características de las anteriores

### 2.3.1 Métricas de tamaño

Se basan en la envergadura del programa para determinar su complejidad. Dentro de este tipo de métricas nos podemos encontrar las basadas en el **número de líneas y las Healstead.**

- **Número de líneas.** Simplemente determinan una relación entre complejidad y líneas de código, incorporando factores que afectan a la complejidad cómo el tipo de lenguaje utilizado o la plataforma en la que se encuentra implantado.

  Esta una métrica muy simple, ya que consiste en contar las líneas del código, pero menos fiable que el resto, ya que presupone que un programa con más líneas que otro más complejo.

  De forma adicional, se pueden incorporar otros factores de medición a parte de las líneas de código, para aumentar la fiabilidad de esta media. Entre estos factores, los más destacados son los siguientes:

- Lenguaje de codificación. Programas codificados en lenguajes diferentes deberán tener complejidades distintas, por ejemplo 100 líneas en ensamblador son mucho más complejas que 100 líneas de VB.
- Omisión de líneas de comentarios. Eliminando estas líneas se aumentará la fiabilidad de la métrica.
- Asignación de pesos diferentes a determinadas partes del código. Las partes del programa de definición o identificación deberán tener menos peso que los algoritmos codificados.

- **Métricas de Halstead.** Métricas de software introducidas por [21] Maurice Howard Halstead en 1977 como parte de su Tratado sobre el establecimiento de una ciencia empírica para el desarrollo de software.

  El objetivo de Halstead es identificar las propiedades medibles del software y las relaciones entre ellos. Esto es similar a la identificación de propiedades medibles de la y las relaciones entre ellos. Por lo tanto, sus métricas no son solo métricas de complejidad.

  Estas métricas se calculan estáticamente a partir del código y su objetivo es el de identificar las propiedades medibles del software y las relaciones entre ellas a partir de las siguientes fórmulas (figura 18):

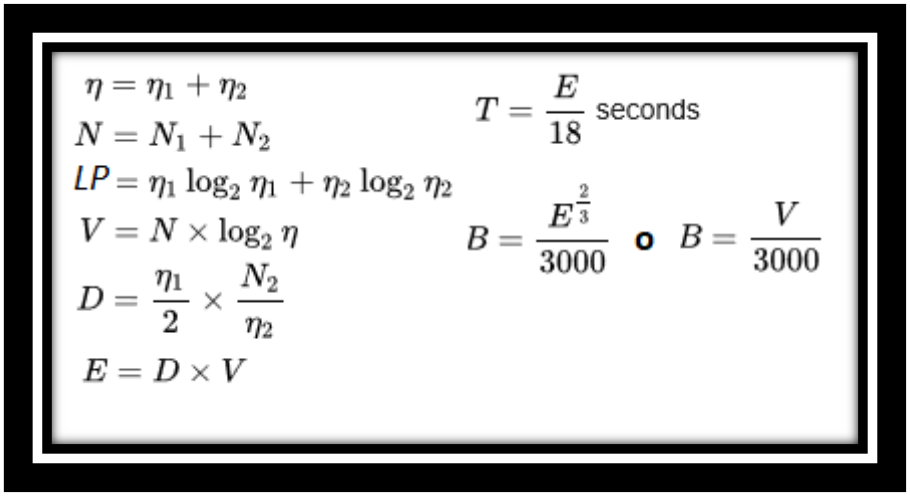



*Figura 18.Formula Halstead*

Los datos que se deducen del código se relacionan en la tabla 17.

| Campo | Descripción |
|---|---|
| **n1** | Número de operadores distintos |
| **n2** | Numero de variables/operandos distintos |
| **N1** | Número total de operadores que se utilizan en el programa |
| **N2** | Número total de variables/operandos utilizados en el programa |

*Tabla 17. Campos entrada Healstead*

En la tabla 18 se describen los datos calculados y su fórmula:

| Campo | Descripción | Fórmula de cálculo |
|---|---|---|
| **N** | Resultado operadores (vocabulario) | n=n1+n2 |
| **N** | Longitud del programa | N=N1+N2 |
| **LE** | Longitud estimada | LE = n1*log2n1+n2*log2n2 |
| **V** | Volumen | V=N*log2n |
| **D** | Dificultad | D=n1/2*N2/n2 |
| **E** | Esfuerzo estimado | E=D*V |
| **T** | Tiempo requerido (seg) | T=E/18 |
| **B** | Número estimado de bug | B=(E^2/3) / 3000 |
| **B (otra posibilidad)** | Número estimado de bug | B=V / 3000 |

*Tabla 18. Campos salida y fórmulas Healstead*

A continuación, un ejemplo sencillo de esta métrica:

Código para analizar:

```
Sub Obtener_elementos_JCL(nom_fichero As String)
   Dim var2_Entrada, var2_Salida, var2_Buscar_disp As Boolean
   fichero = FreeFile
   var2_Buscar_disp=true
   Open nom_fichero For Input As fichero
    num_Linea = 0
    Do While Not EOF(fichero)     ' tratamos hasta el final
        var2_Buscar_disp =false
        num_Linea = num_Linea + 1
        texto = Leer_Sentencia(fichero)
       Msgbox "Buscar_ficha_extendida"
        If Trim(texto) = "//" Then
            Exit Do
        End If
        Msgbox  "Buscar_etiquetas"
        Msgbox  "Buscar_librerias_ficheros"
    Loop

    'caso especial de último paso sin ficheros
        If Trim(texto) = "//" Then
            Msgbox "hola"
        End If
    Close #fichero
    Close #fichero2
```

```
End Sub
```

Datos de entrada. Se relacionan en la tabla 19.

| Operador | Número |
|---|---|
| **If – end if** | 2 |
| **Exit do** | 2 |
| Msgbox | 4 |
| **=** | 5 |
| **Close** | 2 |
| **Open** | 1 |
| **Do while** | 1 |
| **n1=7** | N1=17 |

*Tabla 19. Ejemplo Healstead (campos entrada 1)*

Datos de salida. Se relacionan en la tabla 20.

| Variables/Operandos | Número |
|---|---|
| var2_Entrada | 1 |
| var2_Salida | 1 |
| var2_Buscar_disp | 3 |
| fichero | 5 |
| nom_fichero | 2 |
| num_Linea | 3 |
| // | 2 |
| EOF(fichero) | 1 |
| **n2=8** | N2=18 |

*Tabla 20. Ejemplo Healstead (campos entrada 2)*

Los resultados se relacionan en la tabla 21:

| Campo | Descripción | Resultados |
|---|---|---|
| **n** | Resultado operadores (vocabulario) | **15** |
| **N** | Resultado N | **35** |

| | | |
|---|---|---|
| **LE** | Longitud estimada | **44** |
| **V** | Volumen | **137** |
| **D** | Dificultad | **8** |
| **E** | Esfuerzo estimado | **1.077** |
| **T** | Tiempo requerido (seg) | **60** |
| **B** | Número estimado de bug | **0,035** |
| **B (otra posibilidad)** | Número estimado de bug | **0,046** |

*Tabla 21. Ejemplo Healstead (RESULTADOS)*

### 2.3.2 Métricas de estructura y flujo de datos

En este tipo de métricas se utilizan otros factores más complejos aparte del tamaño. Algunos de estos nuevos factores son:

- El intervalo al hacer referencias a datos. A más distancia entre las referencias a datos mayor complejidad.
- Par de uso segmento-global. A más uso de variables globales, más probabilidad de error.
- Medida [22] Q de Chapin. Asignación de pesos diferentes a tipologías de variables, dividiendo éstas en 4 tipologías dependiendo de su variación de valor, tipo de datos de E/S, etc, dentro de un segmento (las tipologías se clasifican con las letras P, M, C y T).

### 2.3.3 Métricas de flujo de control

Se basan en el cálculo de los distintos caminos que se puedan seguir en la ejecución de un determinado código [23], siendo más complejo el código a mayor número de caminos.

El número de caminos se determina con el análisis de las sentencias condicionales y se llamará número o complejidad ciclomática.

La Complejidad Ciclomática que fue desarrollada originalmente por Thomas McCabe [24] es una métrica del software en ingeniería del software que proporciona una medición cuantitativa de la complejidad lógica de un programa.

Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.

Se establecen 4 niveles de complejidad, que corresponden con la siguiente clasificación por nivels de la tabla 22.

| **Complejidad Ciclomática** | **Evaluación del Riesgo** |
|---|---|

| | |
|---|---|
| De 1 a 10 | Programa Simple, sin mucho riesgo |
| De 11 a 20 | Más complejo, riesgo moderado |
| De 21 a 50 | Complejo, Programa de alto riesgo |
| Más de 50 | Programa no testeable, Muy alto riesgo |

*Tabla 22. Niveles de complejidad ciclomática*

### 2.3.4 Herramientas para medir la calidad software

En la tabla 23, se muestra una relación de herramientas para medir la calidad del software [25] y una breve descripción de sus funcionalidades y funcionamiento:

| Herramienta | Descripción |
|---|---|
| **Check Style** | Analizador estático de código JAVA, para verificar reglas predefinidas cómo estándares (ejemplo: cabeceras, paquetes, etc). Es una herramienta con licencia GNU Lesser General Public License Version 2.1 y la página oficina Página oficial: http://checkstyle.sourceforge.net/. |
| **Google CodePro Analytix** | Esta herramienta ofrece un entorno de trabajo para evaluar el código, ofreciendo métricas de análisis de dependencias, cobertura de código y también ofrece generación de pruebas unitárias.<br>Disponible como plugin de Eclipse. Página oficial: http://code.google.com/intl/es-ES/javadevtools/codepro/doc/index.html.<br>Herramienta gratuíta. |
| **PMD** | Es un analizador que a través de reglas es capaz de identificar problemas que se encuentren en el mismo código, como por ejemplo duplicidad, código no utilizado, alta complejidad de condiciones o bucles, etc.<br>Se encuentra disponible para Java, JavaScript y xsl entre otros.<br>Su Página oficial: http://pmd.sourceforge.net/. |
| **Simian** | Es una solución para la detección de código duplicado.<br>Lenguajes que soporta: Java, C#, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML y Visual Basic.<br>Página oficial: http://www.redhillconsulting.com.au/products/simian/.<br>La licencia es libre si su uso está destinado a proyectos OpenSource. |
| **SONAR** | Herramienta libre muy potente que permite analizar y gestionar la calidad del código a través de reglas y parametrizaciones configurables.<br>Trabaja, principalmente, para Java. Aunque da soporte, gracias a la amplia librería de plugins (algunos de pago), a los |

| | |
|---|---|
| | siguientes lenguajes: ABAP, C, Cobol, C#, Delphi/Pascal, Flex/ActionScript, Groovy, JavaScript, Natural, PHP, PL/SQL, Visual Basic 6, Web y XML.<br>Página oficial: http://www.sonarsource.org. La licencia es: LGPL. |

*Tabla 23. Herramientas medición calidad*

La herramienta de SONAR [26] se encuentra muy extendida en entornos productivos Mainframes para medir la calidad del código COBOL.

En SONAR se puede establecer un nivel de calidad mínimo, asignando un valor cualitativo, configurando que sentencias, propiedades, etc. del código irán degradando la calidad de éste, como por ejemplo el tamaño, ciertas sentencias prohibidas, etc.

Una vez analizado un programa, en la pantalla principal nos aparecerán un resumen de métricas, tales como; líneas de codificación, porcentaje de documentación del código (midiendo % de líneas de comentarios con respecto al código), cumplimiento de reglas (sentencias prohibidas o limitadas en uso) y complejidad. Ejemplo de información en el panel principal en la figura 19:


*Figura 19. Panel principal de control en la herramienta Sonar*

En el ejemplo anterior, el programa ya había sido analizado, por lo que además de las métricas habituales, también muestra el desvío con respecto a anteriores revisiones (Ente paréntesis el porcentaje de mejora o empeora, color rojo indica que se ha empeorado respecto al último análisis y verde que se ha mejorado).

Esta herramienta está basada en métodos de Contador de líneas, incluyendo factores para aumentar la complejidad.

Aparte incluye reglas básicas en condiciones o sentencias prohibidas o limitadas para determinar si supera o no el mínimo de calidad.

## *2.5 Paradigma de la prueba Unitaria*

El objetivo de una prueba [27] es el de validar el código que ha sido desarrollado y una prueba unitaria consiste en la validación de "unidades de código", definiendo esto como la unidad de código testeable más pequeña de la aplicación, que suele corresponder son módulos o programas.

Desde el modelo CMMI se establecen áreas de procesos de validación donde se incluyen prácticas específicas que permiten la organización y preparación de la validación del software a través de un metamodelo de validación [28] . Estas prácticas corresonden con:

- SP 1.1. Seleccionar los productos a validar.
- SP 1.2. Establecer el entorno de validación.
- SP 1.3 Establecer los procedimientos y criterios de validación

En la figura 20 se puede observar la ubicación que ocupa la prueba unitaria dentro de todas las tipologías de pruebas, que no solamente se utilizarán para probar el sistema, sino que también serán utilizadas como vía documental de la aplicación, por lo que es necesario verificar todas las pruebas unitarias para que el software pueda evolucionar de entorno (entorno de desarrollo, preproducción, aceptación y producción).

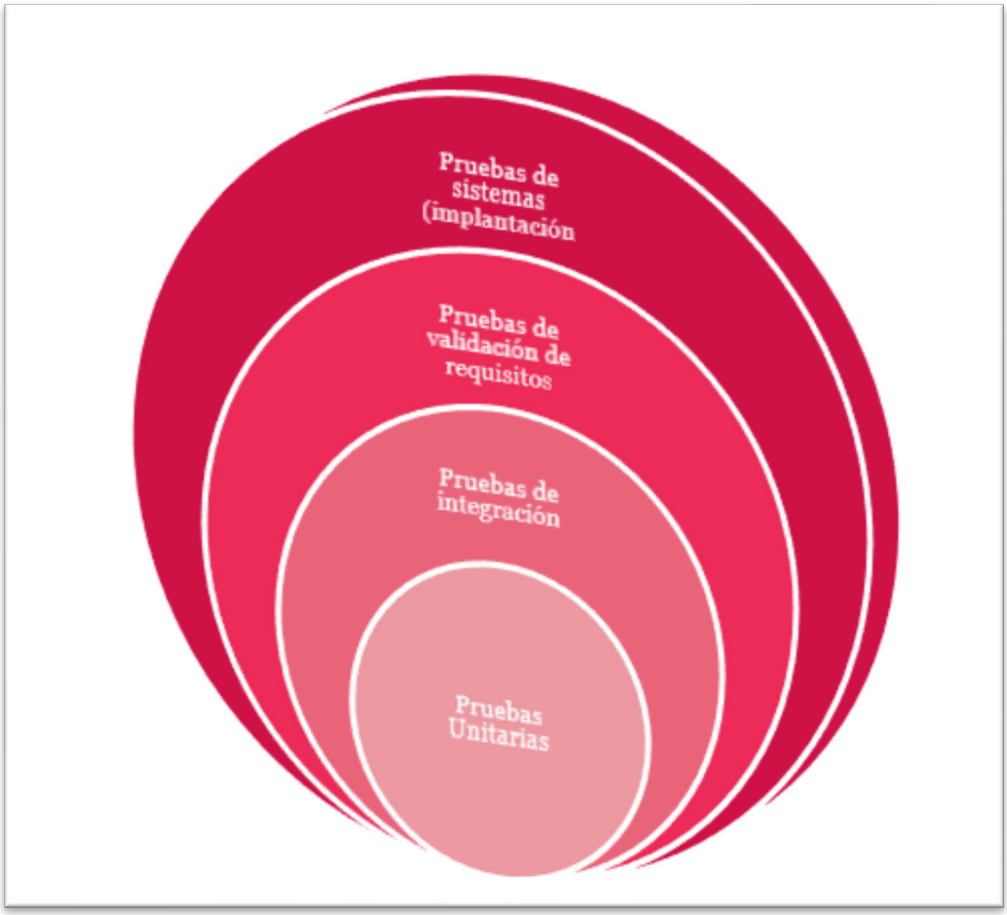



*Figura 20. Tipos estándar de pruebas*

Las pruebas unitarias se centran en el proceso de verificación de la menor unidad del diseño del software (un componente de software o módulo). Las pruebas unitarias suelen ser diseñadas por los programadores, con el objetivo de asegurar el código que han escrito y se podrían definir las siguientes características que deberían tener asociadas para considerar como pruebas aceptables:

- Pruebas independientes. Es aconsejable definir pruebas independientes para no depender de elementos externos.
- Las pruebas se deben definir para asegurar que el código funciona según lo definido.
- Las pruebas también deben incluirse dentro de la documentación del proyecto.
- A mayor complejidad de las pruebas, menor riesgo de fallas del software.
- En desarrollos medianos o grandes, no es viable y sería muy costoso realizar pruebas de todo el código desarrollado, por lo que se deben realizar planteamientos de pruebas inteligentes para minimizar riesgos.
- Debe existir al menos una prueba para cada uno de los requisitos o funcionalidades definidos en el proyecto.
- Deben ser planificados: Actuar, Planear, Ejecutar y Chequear [29].

Los beneficios esperados son los siguientes:

- Minimizar los riesgos de fallas en producción, detectando éstas en las fases previas a la subida.
- Reutilización de software, con la utilización de pruebas de regresión.
- Facilitar las pruebas de integración, al ir más depurado el software unitario.
- Aislamiento de problemas, para no impactar en el resto del software y para facilitar su resolución.

Tal como se ha comentado en algún punto, las pruebas tienen limitaciones y no podemos esperar que se detecten el 100% de los problemas que puedan surgir una vez implantado el software, como por ejemplo en temas de rendimiento, elementos e o fallas ocultas no detectadas en fases previas.

Los errores más comunes detectados en estas pruebas son los siguientes:

- Falta de inicialización de datos.
- Problemas de comparación entre tipos distintos de datos.
- Problemas de movimiento o cálculo con tipos de datos distintos.
- Falta de previsión en el tamaño de variables o estructuras.
- Uso incorrecto de variables.
- Finalización incorrecta de estructuras condicionales.
- Mala construcción en bucles.

En la tabla 24 se realiza un resumen de las reglas básicas que habría que tener en cuenta a la hora de iniciar el proceso de pruebas:

| Regla | Descripción |
|---|---|
| Diseño previo | Es recomendable utilizar técnicas TDD, en las cuales se recomienda el diseño y definición de las pruebas al inicio del desarrollo de proyecto |
| Objetivo principal | El objetivo de la prueba es buscar errores, no que ésta finalice de forma correcta. |
| Nada es obvio | Los fallos de Software suelen producirse por no realizar pruebas del código más básico y en el que se presupone que nunca existirán fallas. |
| Pruebas en vacío | Siempre se debe probar el sistema sin datos, ya que en un porcentaje muy elevado esta casuística suele provocar errores ya sea por falta de datos en todo el sistema o en parte. |
| Problemas de acoplamiento | Antes de implantar el software, éste debe ser probado con los sistemas o partes externas ya existentes en la plataforma dónde vaya a implantarse. |
| Probar requisitos funcionales | Aunque parezca obvio, se deben realizar pruebas específicas para validar los requisitos y funcionalidades definidas en el proyecto, ya que muchas veces el software final NO cumple con las expectativas del cliente y se debe malas interpretaciones. |
| No olvidar los requisitos NO funcionales | No se deben olvidar los requisitos no funcionales, tales como el rendimiento, usabilidad o mantenibilidad entre otros. |

*Tabla 24. Recomendaciones pruebas*

### 2.5.1 Uso de FRAMEWORKS en pruebas unitarias

Existen implementaciones de frameworks de pruebas unitarias en la mayoría de los lenguajes. Es marcos de trabajo, facilitan la realización y reutilización de las pruebas. Podemos destacar los siguientes framewoks:

- Lenguaje Java. Nos encontramos el JUnit que es un framework open source muy estandarizado.**Junit** [30] es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. JUnit es un Framework Open Source para la automatización de las pruebas, tanto pruebas unitarias como de integración, en aplicaciones desarrolladas en Java. El Framework proporciona herramientas, clases y métodos que facilitan la tarea de realizar pruebas en el sistema y así asegurar su consistencia y funcionalidad. También existen otros generadores automáticos de

prueba a partir de código java tal cómo GeCap [31].

- Mundo .net. Tenemos [32] NUnit, una framework open source de pruebas para Microsoft .net y mono. Proporciona una consola al programador que le permite definir y ejecutar lotes de prueba. El codificador a través de un motor de pruebas es capaz de cargar, explorar y ejecutar dichas pruebas.

- Mundo mainfranes [33]. Para poder realizar las pruebas unitarias en esta arquitectura se definen varias herramientas, entre las cuales se pueden destacar las siguientes:
    - Xpediter/CICS: herramienta utilizada para poder ejecutar los programas, parametrizarlos y depurarlos con fin de encontrar los errores. Todos los abend podrán ser consultados con la herramienta Abend-AID (también disponible).
    - Abend-AID: herramienta para consultar todos los Bugs encontrados.
    - File-AID: herramienta para la generación de datos de prueba.
    - Xpediter/Code Coverage: comprobará la cobertura de las pruebas unitarias.

Existen otros enfoques generativos para la realización de pruebas unitarias, como por ejemplo KDM2xUnit (Knowledge Discovery Metamodel-based Unit Test Cases Generation) [34] que permite generar conjuntos de pruebas que se pueden asociar a framework xUnit, realizando transformaciones y metamodelos de descubrimiento de conocimiento (KDM) o herramientas como Sulu [35], basadas en lenguajes de programación diseñados para automatizar pruebas unitarias, generando y automatizando su ejecución.

### 2.5.2 Otras actividades de detección de errores diferentes a las Pruebas

Aunque la prueba es una de las actividades clave de control de calidad, hay más acciones adicionales que se podrían adoptar para asegurar que la calidad del producto cumple con algunos objetivos, este tipo de acciones y técnicas, se basan en inspeccionar el código y aunque existe escepticismo entre las empresas a la hora de su utilización, se ha demostrado que funcionan:

- Ahorre de costes al encontrar los errores en etapas tempranas del desarrollo:
    - Ahorro a la hora de corregirlos.
    - Ahorro en la repetición de pruebas unitarias, de caja negra, etc. si éstos son localizados en primeras etapas.
- Mejora del aprendizaje en los desarrolladores. Las personas que revisan el código también están aprendiendo de forma paralela.
- Estudios han demostrado que entre un 50%-70% son eficaces y se localian errores en la lógica y el diseño de programas

Entre las técnicas más destacadas se encuentran las siguientes:

- Inspección formal del código.
- Tutorización de prueba.
- Revisión informal de código
- Par de programación.

Inspección formal del código.

Se crea un equipo específico para la inspección de código [36], compuesto por 4 personas de forma estándar, aunque podrá variar dependiendo del tamaño del proyecto. Entre las personas que incorporan el equipo debe existir un moderador, que será un experto programador y en ningún caso debe ser el autor del software. Las actividades son:

1. El programador describe, sentencia a sentencia la lógica del software. El resto de oyentes, podrán realizar preguntas con el fin de determinar si existen errores.
2. El programa se analiza en relación con las listas de comprobación de errores históricos comunes.
3. Al finalizar la sesión, el programador recibe una lista de errores encontrados para solventarlos.

La lista de errores no sólo es utilizada por el desarrollador con el fin de corregirlos, sino que también se utiliza para verificar si la lista de comprobación de error podría mejorar resultados.

Estas sesiones de revisión deben ser dinámicas y su duración oscilará entre los 90-120 minutos, tal como se puede observar en la figura 21.

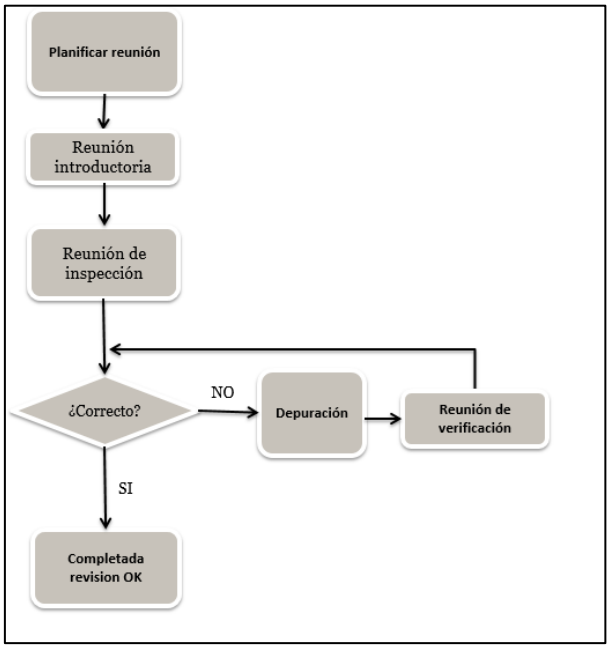



*Figura 21.Diagrama de pasos para una inspección formal de código*

- Planificar reunión. Comprobar materiales y convocar reunion con todos los participantes.
- Reunión introductoria. El autor presenta los materiales, los moderadores fijan las expectativas, la reunión de inspección está programada
- Reunión de inspección. Revisar materiales, registrar defectos, recoger métricas.
- Depuración. El autor implementa los cambios acordados y se programa una reunión de verificación.
- Reunión de verificación. Para comprobar los defectos se corrigen adecuadamente.

Tutorización de prueba

Similar a la "inspecciones formales de código" en composición de equipos y duración, pero en lugar de realizar una lectura del código, se utilizan listas de control, para realizar una especie de juego ente los participantes a los que se les asignarán casos de prueba para verificar la lógica del programa, que se supervisará en una pizarra en común.

Los casos de prueba no deben ser un conjunto completo de casos de prueba, sobre todo porque cada ejecución mental de un caso de prueba sería muy compleja. La idea es que se le cuestione al programador las decisiones tomadas.

Además del autor del software y un moderador, también se incorporan dos nuevas funciones para esta técnica:

- Probador, que es el responsable de guiar la ejecución de los casos de prueba.
- Secretario que escribe todos los errores encontrados.

Revisión informal de código

Las anteriores técnicas son muy formales y requieren mucho esfuerzo para llevarlas a cabo, además del gasto de recursos. Esta técnica sin embargo es informal y consiste en una revisión entre el desarrollador del código y un supervisor, sin establecimiento de normas.

El desarrollador, debe ir explicando el funcionamiento de su código al supervisor, y si éste tiene dudas o detecta algún error, lo discute con el desarrollador y toma las notas correspondientes.

Al final de la revisión, se entrega al desarrollador un informe con los errores localizados.

Estas etapas se pueden ver en la figura 22:

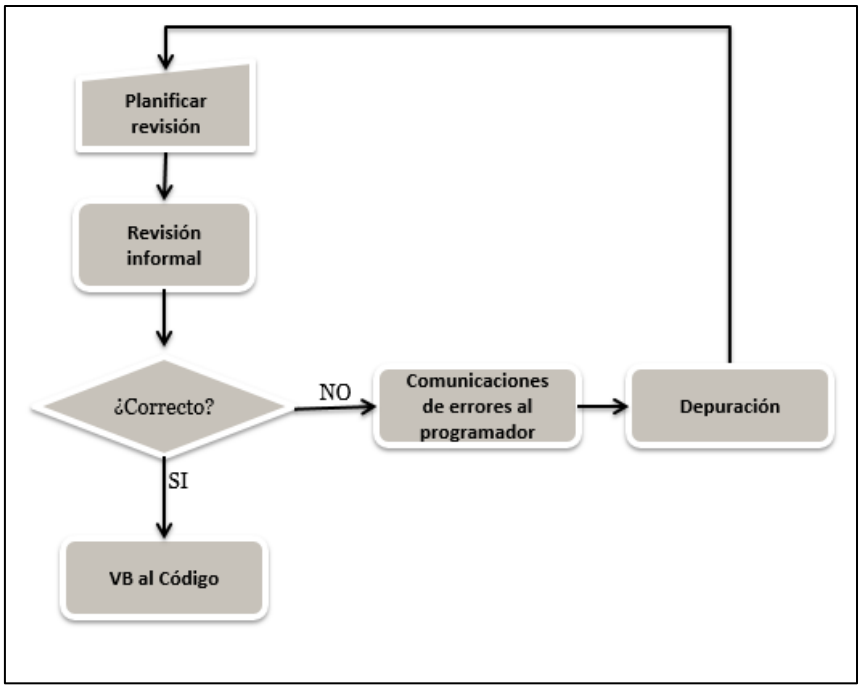



*Figura 22. Pasos para realizar una inspección informal de código*

- Planificar revisión. Se acuerda fecha y duración de la reunión entre el desarrollador y supervisor se planifica fecha y duración de la reunión.
- Revisión informal. Se realiza una revisión informal del código ente ambos componentes.
- Comunicación de errores. Si se han detectado errores u otros problemas, el supervisor realiza un informe y se lo entrega al programador, para depurar el código.
- VB al código. Si no se han detectado problemas, se da por finalizada la revisión del código.

Existe otra variante a la revisión informal de código, que se trata de realizar la revisión pero a distancia, en la cual el desarrollador le envía al revisor el código y archivos asociados al software (o compartir en git u otras plataformas), para que éste los analice.

La principal ventaja con respecto con la anterior es que se puede trabajar a distancia. Además, mediante el uso de esta técnica es muy fácil permitir que varios revisores para revisar el código en paralelo.

La desventaja principal es que se necesita más tiempo, ya que por lo general requiere diferentes interacciones, esto podría ser especialmente costoso si las personas se encuentran en diferentes zonas horarias.

El siguiente diagrama (figura 23), se describe este proceso :

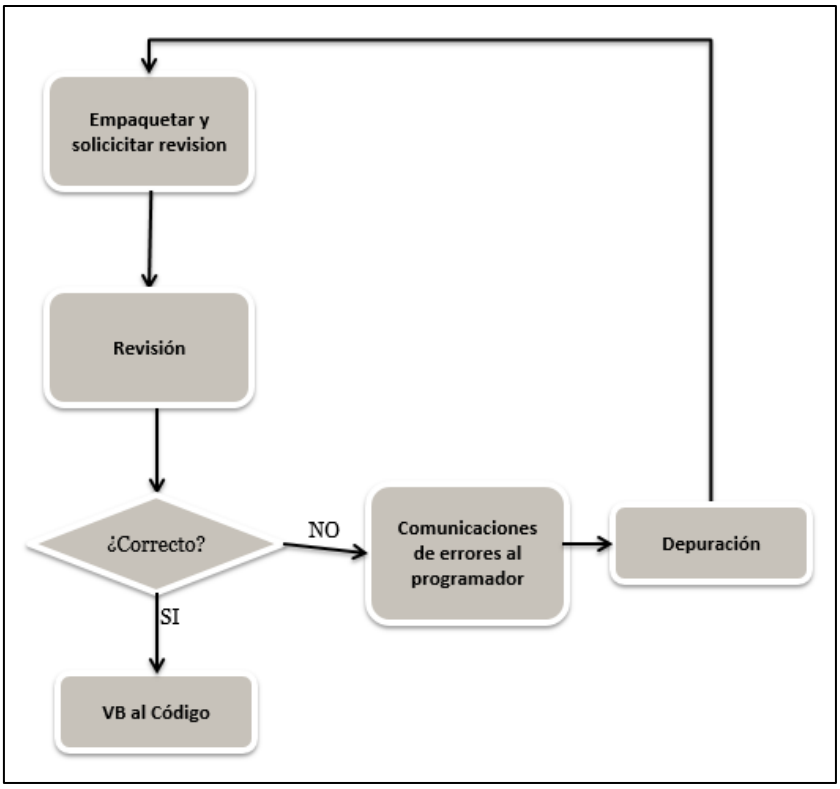



*Figura 23. Pasos para una inspección informal de código a distancia*

- Empaquetar y solicitar revisión. El desarrollador prepara un paquete del software y se lo remite al revisor, con todos los cambios del software.
- Revisión. El supervisor revisa el paquete de software recibido.
- Depuración. El codificador depura el software con los comentarios recibidos y vuelve a solicitar una nueva revisión cuando lo tenga preparado.
- Completado. VB al software.

Par de programación

Par programación, es un proceso de desarrollo que incorpora la revisión de código continua en el propio proceso de desarrollo. Consiste en dos desarrolladores que escriben código en un solo terminal con sólo un desarrollador a escribir a la vez y de forma libre continua discusión y revisión.

## *2.6 Situación actual y estadísticas de utilización de lenguajes*

Según estadísticas realizadas [37] en la última década sobre la utilización de lenguajes de programación, los lenguajes VB, JAVA y COBOL han estado ocupando las 20 primeras posiciones, incluso lenguaje COBOL que se podría considerar que podría ir perdiendo posiciones, las ha ido ganando en lugar de perderlas, cómo por ejemplo en el ranking del año 2013, tal cómo se puede observar la tabla 25.

| Position Nov 2013 | Position Nov 2012 | Delta in Position | Programming Language | Ratings Nov 2013 | Delta Nov 2012 | Status |
|---|---|---|---|---|---|---|
| 1 | 1 | = | C | 18.155% | -1.07% | A |
| 2 | 2 | = | Java | 16.521% | -0.93% | A |
| 3 | 3 | = | Objective-C | 9.406% | -0.98% | A |
| 4 | 4 | = | C++ | 8.369% | -1.33% | A |
| 5 | 6 | ↑ | C# | 6.024% | +0.43% | A |
| 6 | 5 | ↓ | PHP | 5.379% | -0.35% | A |
| 7 | 7 | = | (Visual) Basic | 4.396% | -0.64% | A |
| 8 | 8 | = | Python | 3.110% | -0.95% | A |
| 9 | 23 | ↑↑↑↑↑↑↑↑↑↑ | Transact-SQL | 2.521% | +2.05% | A |
| 10 | 11 | ↑ | JavaScript | 2.050% | +0.77% | A |
| 11 | 15 | ↑↑↑↑ | Visual Basic .NET | 1.969% | +1.20% | A |
| 12 | 9 | ↓↓↓ | Perl | 1.521% | -0.66% | A |
| 13 | 10 | ↓↓↓ | Ruby | 1.303% | -0.44% | A |
| 14 | 14 | = | Pascal | 0.715% | -0.17% | A |
| 15 | 13 | ↓↓ | Lisp | 0.706% | -0.25% | A |
| 16 | 19 | ↑↑↑ | MATLAB | 0.656% | +0.04% | B |
| 17 | 12 | ↓↓↓↓↓ | Delphi/Object Pascal | 0.649% | -0.35% | A- |
| 18 | 17 | ↓ | PL/SQL | 0.605% | -0.03% | A- |
| 19 | 24 | ↑↑↑↑↑ | COBOL | 0.585% | +0.11% | B |
| 20 | 20 | = | Assembly | 0.532% | -0.05% | B |

*Tabla 25. Ranking años 2013 y 2012 según TIOB*

En junio de 2016, según información de TIOBE (compañía dedicada a la calidad del Software), JAVA sería el lenguaje más utilizado, el VB se encontraría en la posición 11 y el cobol se encontraría en la posición 20 con un porcentaje algo superior del 1%.

| Ranking | Lenguaje | Porcentaje uso |
|---|---|---|
| 1 | Java | 20.794% |
| 2 | C | 12.376% |
| 3 | C++ | 6.199% |
| 4 | Python | 3.900% |
| 5 | C# | 3.786% |
| 6 | PHP | 3.227% |
| 7 | JavaScript | 2.583% |
| 8 | Perl | 2.395% |
| 9 | Visual Basic .NET | 2.353% |
| 10 | Ruby | 2.336% |
| 11 | Visual Basic | 2.254% |
| 12 | Ensamblador | 2.119% |

| | | |
|---|---|---|
| **13** | Delphi/Object Pascal | 1.939% |
| **14** | Swift | 1.831% |
| **15** | Objective | C |
| **16** | R | 1.540% |
| **17** | MATLAB | 1.447% |
| **18** | PL/SQL | 1.346% |
| **19** | D | 1.063% |
| **20** | COBOL | 1.048% |

*Tabla 26. Ranking año 2016 según TIOBE*

Para las estadísticas de la tabla 26 [38], se ha tenido en cuenta todo el espectro de entornos de desarrollo, pero sí los categorizamos, COBOL se situaría en la posición número 2 de lenguajes de programación de otras categorías distintas a script y a generar-purpose, tal como se muestra en la tabla 27:

Language category: **general-purpose** [1]
48 entries.

| Rank | Name | Share |
|---|---|---|
| 1 | C | 26.623% |
| 2 | Java | 22.182% |
| 3 | Objective-C | 12.401% |
| 4 | C++ | 10.202% |
| 5 | Basic | 8.223% |
| 6 | C# | 4.926% |
| 7 | Pascal | 1.906% |
| 8 | Ada | 1.688% |
| 9 | Fortran | 1.201% |
| 10 | Forth | 1.148% |
| 11 | D | 0.955% |

Language category: **script** [1]
49 entries.

| Rank | Name | Share |
|---|---|---|
| 1 | PHP | 19.801% |
| 2 | Python | 16.458% |
| 3 | Perl | 11.545% |
| 4 | Ruby | 8.628% |
| 5 | JavaScript | 5.399% |
| 6 | R | 4.898% |
| 7 | Lisp/Scheme | 3.960% |
| 8 | MATLAB | 3.488% |
| 9 | Bourne shell | 2.757% |
| 10 | APL | 2.316% |
| 11 | NXT-G | 1.871% |

Language category: **other** [1]
26 entries.

| Rank | Name | Share |
|---|---|---|
| 1 | Logo | 14.778% |
| 2 | COBOL | 14.597% |
| 3 | Prolog | 10.099% |
| 4 | PL/SQL | 8.939% |
| 5 | SAS | 8.619% |
| 6 | LabView | 8.317% |
| 7 | ABAP | 6.482% |
| 8 | Transact-SQL | 2.837% |
| 9 | Focus | 2.728% |
| 10 | XSLT | 2.713% |
| 11 | YACC | 2.550% |

*Tabla 27. Ranking por categorización de uso*

Pero la gran mayoría de encuestas [39] desde hace ya 2 décadas siguen arrojando datos poco alentadores sobre la continuidad del COBOL y el VB que van perdiendo posiciones, tal como se puede observar en la tabla 28 el ranking de TIOBE de noviembre de 2018, lo que provoca que nuevas generaciones no quieran aprender estos lenguajes. Esto está provocando una falta de programadores con respecto a la demanda actual que sigue existiendo y existe una previsión de que así seguirá siendo a corto-medio plazo.

| nov-18 | nov-17 | Evolución | Language | Ratings | Change |
|---|---|---|---|---|---|
| **1** | 1 | | Java | 16746% | +3.51% |
| **2** | 2 | | C | 14396% | +5.10% |
| **3** | 3 | | C++ | 8282% | +2.94% |
| **4** | 4 | | Python | 7683% | +3.20% |
| **5** | 7 | ▲ | Visual Basic .NET | 6490% | +3.58% |
| **6** | 5 | ▼ | C# | 3952% | +0.94% |
| **7** | 6 | ▼ | JavaScript | 2655% | -0.32% |
| **8** | 8 | | PHP | 2376% | +0.48% |
| **9** | - | ⇈ | SQL | 1844% | +1.84% |
| **10** | 14 | ⇈ | Go | 1495% | -0.07% |
| **11** | 19 | ⇈ | Objective-C | 1476% | +0.06% |
| **12** | 20 | ⇈ | Swift | 1455% | +0.07% |
| **13** | 9 | ⇊ | Delphi/ObjectPascal | 1423% | -0.32% |
| **14** | 11 | ▼ | R | 1407% | -0.20% |
| **15** | 10 | ⇊ | Assembly language | 1108% | -0.61% |
| **16** | 13 | ▼ | Ruby | 1091% | -0.50% |
| **17** | 12 | ⇊ | MATLAB | 1030% | -0.57% |
| **18** | 15 | ▼ | Perl | 1001% | -0.56% |
| **19** | 18 | ▼ | PL/SQL | 1000% | -0.45% |
| **20** | 17 | ▼ | Visual Basic | 0.854% | -0.63% |

*Tabla 28. Estadísticas uso de lenguajes noviembre 2018*

# CAPÍTULO 3

# 3. Solución del problema

En este capítulo se dará solución a la problemática descrita en la sección 1.3 del documento, utilizando diferentes técnicas y metodologías para la gestión de proyectos y su desarrollo.

## *3.1 Datos generales*

Fecha de inicio: 01/02/2019

Código: PROYECTO

Título: El título

CDD: (2804) Secretaría del Consejo de Administración

Solicitante: Proyecto TFM.

Responsable del desarrollo/Destinatario:

- Abel García Sánchez

## *3.2 Objetivo del proyecto*

El objeto de este proyecto es el crear unas herramientas que solventen los problemas indicados en el apartado 1.3 (problemática y necesidad de herramientas) sobre la falta de conocimiento en estos sistemas Mainframes, además de aprovechar los procesos automáticos de generación para mejorar los tiempos en la obtención del software (tanto código, como documental) y aumentar la calidad del software generado a través de la mejora de las pruebas unitarias.

Estas herramientas serán desarrolladas partiendo de paradigmas de generación automática de código, análisis semántico de código para generación de documentación, aplicación de técnicas de MDD/MDA para el desarrollo de metamodelos que permitan aplicar abstracciones en el código JCL y diseñen DFD gráficos de diferentes niveles, para obtener una visión global del funcionamiento del software, etc.

Las funcionalidades que se tratarán son las siguientes:

**Entorno documentación de procesos**

Generación de documentación técnica a través de análisis semántico del código de las cadenas JCL.

Partiendo del código de la cadena que está escrito en JCL, se obtendrá información relativa al software ubicado en la cadena. Esta información se registrará en unas tablas intermedias (previamente definición de su modelo), para su posterior utilización en la generación de diagramas y documentación.

Una vez guardados todos los elementos software de la cadena y sus características, se procederá a la generación de la documentación técnica de alto nivel. El documento técnico de alto nivel contendrá la documentación se generará como sigue:

- Arquitectura de componentes Software (elementos software, tipo, nombre, etc).
- Diagrama de ejecución de la cadena. DFD (Diagrama de Flujo de Datos) con abstracción de nivel 1. Se deberá definir un metamodelo para poder transformar la codificación en JCL en un modelo gráfico con tipología DFD.
- Descripción detallada de los pasos del JCL. Definición más de tallada de los pasos del JCL, indicando interfaces de E/S y descripción del paso.
- Modelo físico de datos. Definición de los formatos de interfaces y de tablas utilizadas en el JOB.

**Entornos de pruebas/análisis de código COBOL.**

Partiendo del código fuente de un programa COBOL se obtendrá información que permita generar un entorno de pruebas (JCL de ejecución, programa principal si fuera necesario y juego de datos de prueba).

Partiendo de un programa fuente en COBOL se realizará un análisis del código que permita obtener un documento analítico del programa con la siguiente información:

- Información sobre el diseño técnico del programa:
    - Comentarios del programa sobre su funcionalidad.
    - Arquitectura de componentes.
    - Diagrama de ejecución.

  - Pseudocódigo del programa.

- Medición de la calidad y complejidad del código a través de métodos empíricos. (Métrica de Halstead, complejidad ciclomática, etc) y plasmar el resultado.

- Detección automática de fallas o posibles problemas potenciales en el programa a través de muestreos (revisión de límites, cobertura, variables no usadas, caminos no utilizados, variables sin inicializar, etc).

- Crear un grafo con el flujo de caminos que puede tener el programa.

- Generación de un entorno del entorno de pruebas para poder ejecutar en desarrollo.

  - Generación del JCL (Job Control Languaje). El JCL sería una especie de 'lanzador' del programa para entorno MAINFRAMES, para permitir la ejecución de las pruebas unitarias. Estudio de métodos de generación directa de código.

  - Generación automática de código COBOL para crear un programa llamador. Estudio y análisis de las diferentes técnicas y métodos de generación automática de código: lenguaje de dominio específico DSL's, paradigma basado en modelos (MDA/MDD), generación a través de modelos, metalenguajes, inyección de código, uso de plantillas, etc

- Generar un juego de casos de pruebas unitarias que deberán ser probados.

  - Casos de pruebas que se deberían incluir:

    - Interfaz E/S
    - Pruebas en vacío.
    - Estructuras de datos locales
    - Límites
    - Caminos independientes.
    - Caminos de errores

El dominio del proyecto está basado en la creación de documentación automática de documentación y de generación programas principales para realizar pruebas de módulos.

La herramienta no estará centrada en la producción de un único producto, sino será como una cadena de montaje donde se generarán productos similares de forma automática según la parametrización que introduzca el usuario.

## *3.3 Metodología y planificación*

Se utilizará la metodología clásica en cascada o por fases, ya que los requisitos quedarán bien definidos en una primera fase de análisis de requisitos y éstos no serán variados.

Este es un modelo en cascada, uno de los primeros paradigmas utilizados en la ingeniería de softweare, y de los más usados, aunque no está exento de críticas [40] y ponen en duda su eficacia en contra de las nuevas metodologías AGILE como Scrum [41].

Pero para el caso de estudio, no existirán entregas parciales, implantaciones, evolutivos, etc., por lo que no es recomendable metodologías AGILE tipo Scrum, en las cuales se definen MPV (Mínimos Productos Viables) y se realizan entregas parciales. Si existieran cambios de requisitos o inclusión de nuevos, el impacto sería muy elevado en este tipo de metodologías en cascada [42]

Las fases del proyecto se describen en la figura 24 [23]:

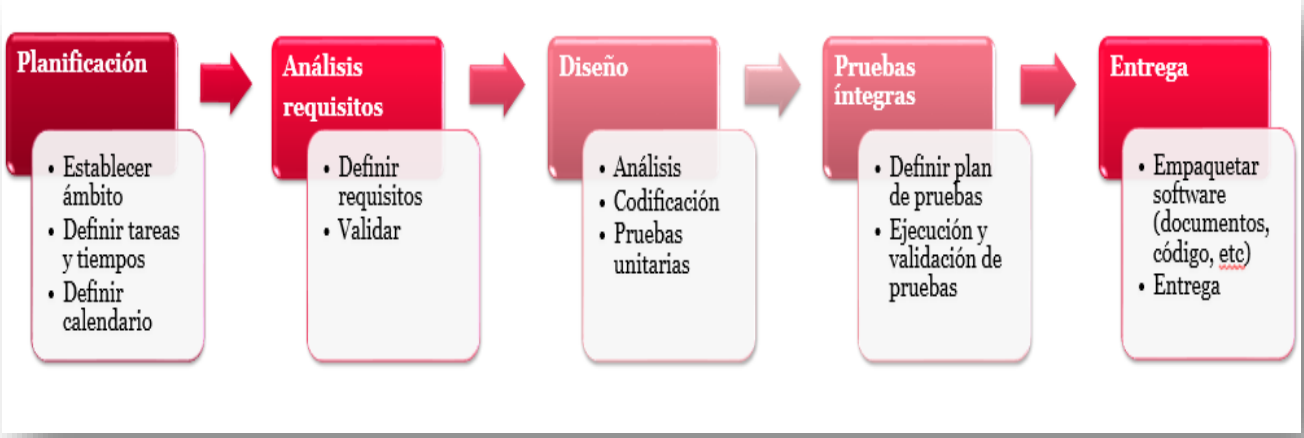



*Figura 24. Metodología de desarrollo clásica: Waterfall*

- **Planificación del proyecto.** En esta fase se deben inventariar todas las tareas necesarias para la consecución del proyecto y su planificación. Para la realización de esta tarea, se han definido 4 PPT de seguimiento (adjuntas en anexos).
- **Análisis de requisitos.** Definición de los requisitos y funcionalidades que debe tener la herramienta. Fase muy importante, ya que estos requisitos y funcionalidades se deben tener en cuenta en las pruebas íntegras, para verificar que se cumple con el objetivo establecido.
- **Diseño, construcción y pruebas unitarias.** Fase que aborda el diseño y la construcción del Software según los requisitos y funcionalidades definidadas, con la incorporación de las pruebas unitarias de cada elemento que se vaya desarrollando.
- **Pruebas de integración.** Pruebas de caja negra, caja blanca, de implantación, etc. Crear un plan de pruebas para poder dar el VB. Revisar el análisis de pruebas desarrollado en calidad del software.
- **Entrega.** En el caso de estudio, la herramienta no se implantará en ninguna plataforma en concreto.

## Planificación del proyecto

Para el correcto desarrollo del proyecto [43] y su seguimiento, se han definido las siguientes tareas, con sus esfuerzos estimados y sus fechas estimadas de inicio y de fin, reflejado en la tabla 29.

| TAREAS | | FECHAS PREVISTAS | |
|---|---|---|---|
| **Tareas previas al inicio** | **Estimación en horas** | **Inicio** | **Fin** |
| Lectura de artículos e investigación sobre temas a tratar en el proyecto | 40 | 15/01/19 | 05/02/19 |
| Realización de propuestas, reuniones con el tutor y selección de los contenidos finales del proyecto | 10 | 17/01/19 | 20/02/19 |
| TOTAL | 50 | **REALES** | 65 |
| **Documentación automática de procesos JCL** | **Horas** | **Inicio** | **Fin** |
| Documentación (análisis, dfd, arquitecturas planteadas, pruebas, etc) | 25 | 26/02/19 | 31/03/19 |
| Definición de estructuras de datos y metamodelos. | 10 | 26/02/19 | 27/02/19 |
| Codificación y pruebas unitarias del parse para JCL (obtención de objetos) | 35 | 28/02/19 | 03/03/19 |
| Codificación y pruebas unitarias generación DFD niv 1 | 15 | 04/03/19 | 07/03/19 |
| Codificación y pruebas unitarias generación DFD niv 2 | 50 | 08/02/19 | 22/03/19 |
| Codificación y pruebas unitarias de la generación del documento técnico | 20 | 23/03/19 | 27/03/19 |
| Preparación de casos de prueba | 15 | 28/03/19 | 31/03/19 |
| **TOTAL PREVISIÓN** | 170 | **REALES** | 180 |
| **Preparación entorno de pruebas COBOL** | **Horas** | **Inicio** | **Fin** |
| Documentación (análisis, dfd, arquitecturas planteadas, pruebas, etc) | 25 | 01/04/19 | 05/05/19 |
| Definición de estructuras de datos y metamodelos. | 10 | 01/04/19 | 03/04/19 |
| Codificación del Parse lenguaje COBOL | 35 | 04/04/19 | 13/04/19 |
| Generación automática de código (programa COBOL y JCl) para entorno de pruebas | 20 | 14/04/19 | 18/04/19 |
| Estimar calidad y complejidad por métodos empíricos | 60 | 19/04/19 | 03/05/19 |
| Refundir toda la información en un documento DT y pruebas íntegras. | 15 | 03/05/19 | 05/05/19 |
| **TOTAL PREVISIÓN** | 165 | **REALES** | |
| **Documentación y otras tareas** | **Horas** | **Inicio** | **Fin** |
| Pruebas íntegras de toda la herramienta y correcciones | 20 | 05/05/19 | 10/05/19 |
| Documentación de la memoria | ~~50~~ 70 | 01/03/19 | ~~20/05/19~~ 20/06/19 |
| Planificación y seguimiento del proyecto | 10 | 26/02/19 | ~~15/06/19~~ |

| | | | |
|---|---|---|---|
| | | | 01/07/19 |
| Preparación de la defensa | 20 | ~~01/06/19~~ 15/06/19 | ~~15/06/19~~ 01/07/19 |
| **TOTAL PREVISIÓN** | ~~80~~ 100 | **REALES** | |
| TOTAL PROYECTO (PREVISIÓN) | ~~435~~ 455 | REALES | |

*Tabla 29. Planificación de tareas y su estimación*

**Planificación tareas.**

-En la figura 25 se muestra la Planificación a fecha 31 de marzo de 2019

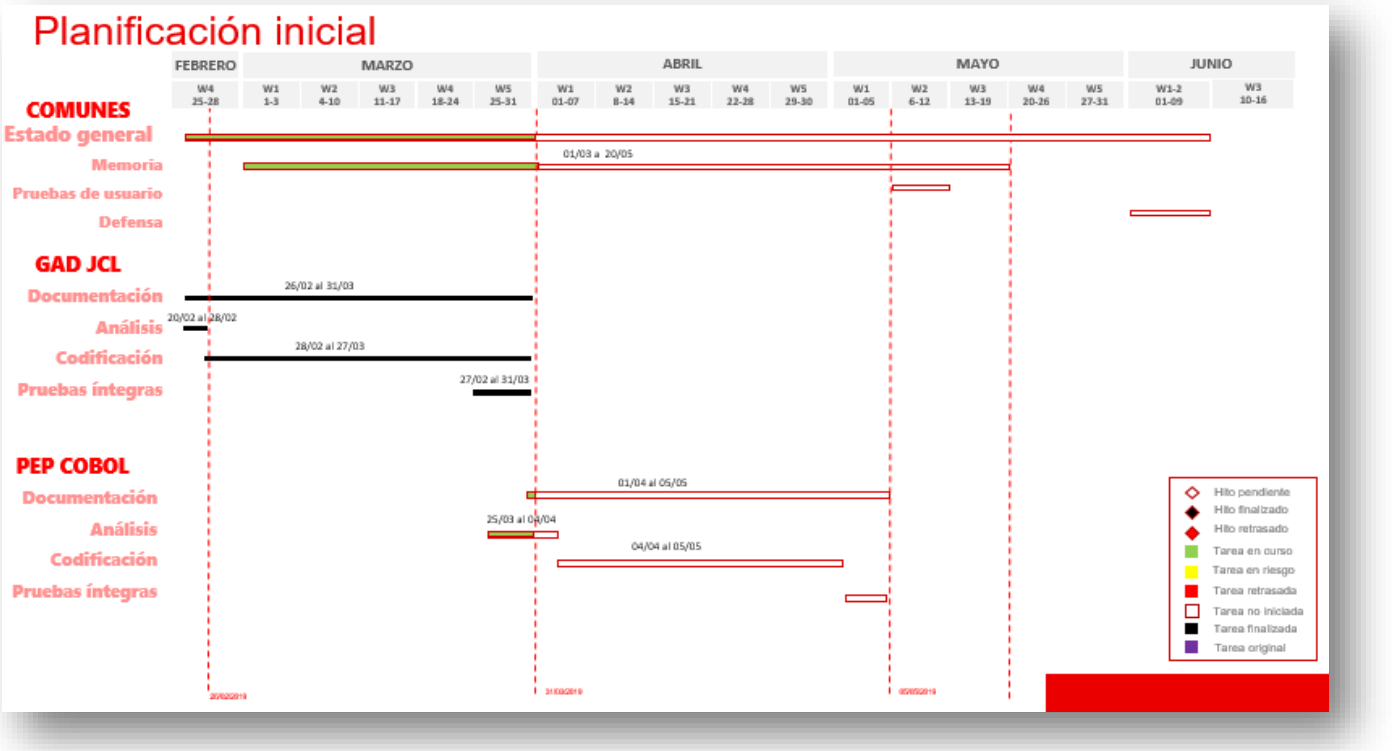



*Figura 25. Planificación por la aplicación proyect del proyecto*

Se define una PPT para realizar el seguimiento del proyecto. Los contenidos de este documento son los siguientes (ver anexo Planificación proyecto):

1. Objetivo del proyecto y situación general del avance
2. Revisión de Tareas/hitos
3. Planificación Inicial y Actualizada
4. Situación de entregables
5. Riesgos detectados

## *3.4 Definición de requisitos*

En este apartado se definirán los requisitos [23] deducidos a partir de la problemática descrita en los apartados anteriores.

En la tabla 30 se enumeran los requisitos que han sido deducidos:

'Definir 3 requisitos principales para cada una de las funcionalidades principales y después subrequisitos en cada una de ellas'

A contuación, se relacionan los subrequisitos de la funcionalidad de pruebas.

| Identificación Requisito | Descripción del requisito | Tipo de requisito |
|---|---|---|
| REQ-001 | Generación automática de documentación técnica sobre procesos JCL. | Funcional |
| REQ-001-01 | Analizador semántico de JCL para obtener la arquitectura de componentes software del JCL | Funcional |
| REQ-001-02 | Creación de un diagrama de flujo de datos de ejecución (nivel1) | Funcional |
| REQ-001-03 | Creación de un diagrama de flujo de datos detallado del JCL (nivel2) | Funcional |
| REQ-001-04 | Generación de un DT del proceso con un formato predefinido. | Funcional |
| REQ-002 | Generación automática de entornos de prueba Mainframes y analizador de código COBOL | Funcional |
| REQ-002-01 | Generación del código COBOL del programa principal para pruebas. | Funcional |
| REQ-002-02 | Generación del entorno de pruebas del lanzador para entornos MAINFRAMES (generación del JCL).. | Funcional |
| REQ-002-03 | Medición de la calidad y complejidad del código a través de métodos empíricos | Funcional |
| REQ-002-04 | Detección automática de fallas o posibles problemas potenciales en el programa a través de muestreos | Funcional |
| REQ-002-05 | Crear un grafo con el flujo de caminos que puede tener el programa. | Funcional |
| REQ-002-06 | Generación automática del DT del programa con el formato predeterminado. | Funcional |
| REQ-003 | Facilidad de uso | No funcional |
| REQ-004 | Manual de usuario | No funcional |
| REQ-005 | Rendimiento óptimo | No funcional |
| REQ-006 | Seguimiento del estado del proceso | No funcional |

*Tabla 30. Definición de requisitos*

***Tabla 6. Definición de requisitos***

Definición detallada de cada requisito:

**Requisito 1 y sub-requisitos asociados en tabla 31.**

| **ID Requisito:** REQ-001 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 10-02-2019 | |
| **Breve descripción** | Generación automática de documentación técnica sobre procesos JCL. | | |
| **Descripción detallada** | Partiendo del código fuente de un proceso de tipología Job Control Language, se debe obtener un documento de tipología técnica donde se detalle la arquitectura de componentes software, un diagrama de ejecución del proceso, un diagrama detallado de flujo de datos y el detalle de cada uno de los pasos que conforman el JCL.<br><br>El resultado de los distintos análisis será trasladado a un documento de diseño técnico del programa. | | |
| **Criterio de aceptación** | Revisión de pruebas integras. | | |
| **Origen e Historia** | Descripción | Fecha | |
| | Añadido | 10-02-2019 | |
| | Aceptada | 05-03-2019 | |

*Tabla 31. Definición requisito REQ-001*

Subrequisito REQ-001-01 definido en tabla 32.

| **ID Requisito:** REQ-001-01 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 15-02-2019 | |
| **Breve descripción** | Obtención de la arquitectura de componentes software del proceso JCL. | | |
| **Descripción detallada** | Realizando un análisis semántico del JCL (un parse) para obtener toda la tipología de componentes software que componen el proceso. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |
| **Origen e Historia** | Descripción | Fecha | |
| | Añadido | 15-02-2019 | |
| | Aceptada | 05-03-2019 | |

*Tabla 32. Definición requisito REQ-001-01*

Subrequisito REQ-001-02 definido en tabla 33.

| **ID Requisito:** REQ-001-02 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 20-02-2019 | |

| | | |
|---|---|---|
| **Breve descripción** | Creación de un diagrama de flujo de datos de ejecución (nivel1 Diagrama de nivel superior) | |
| **Descripción detallada** | Partiendo de la arquitectura de componentes del proceso, se debe generar un diagrama de ejecución del proceso (un nivel de detalle 1). | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | |
| **Origen e Historia** | Descripción | Fecha |
| | Añadido | 22-02-2019 |
| | Aceptada | 05-03-2019 |

*Tabla 33. Definición requisito REQ-001-02*

Subrequisito REQ-001-03 definido en tabla 34.

| **ID Requisito:** REQ-001-03 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 20-02-2019 | |
| **Breve descripción** | Creación de un diagrama de flujo de datos detallado del JCL (nivel 2 Diagrama de detalle o expansión) | | |
| **Descripción detallada** | Partiendo de la arquitectura de componentes del proceso, se debe generar un diagrama de flujo de datos más detallado, con un nivel de detalle 2. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |
| **Origen e Historia** | Descripción | Fecha | |
| | Añadido | 22-02-2019 | |
| | Aceptada | 05-03-2019 | |

*Tabla 34. Definición requisito REQ-001-03*

Subrequisito REQ-001-04 definido en tabla 35.

| **ID Requisito:** REQ-001-04 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 20-02-2019 | |
| **Breve descripción** | Generación de un DT del proceso con un formato predefinido. | | |
| **Descripción detallada** | Se debe generar de forma automática un diseño técnico de proceso JCL que contenga los siguientes apartados:<br>• PROPÓSITO DEL DOCUMENTO<br>• ARQUITECTURA DE COMPONENTES (HW Y SW)<br>• DIAGRAMA DE EJECUCIÓN (DFD NIVEL 1)<br>• DIAGRAMA DETALLADO (DFD NIVEL 2)<br>• DESCRIPCIÓN DETALLADA DE COMPONENTES<br>• DOCUMENTOS RELACIONADOS<br>• ANEXOS | | |

| | | |
|---|---|---|
| | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias e íntegras | |
| **Origen e Historia** | Descripción | Fecha |
| | Añadido | 24-02-2019 |
| | Aceptada | 05-03-2019 |

*Tabla 35. Definición requisito REQ-001-04*

**Requisito 2 y sub-requisitos asociados en tabla 36.**

| **ID Requisito:** REQ-002 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 20-02-2019 | |
| **Breve descripción** | Generación automática de entornos de prueba Mainframes y analizador de código COBOL. | | |
| **Descripción detallada** | Requisito principal para la obtención de un entorno de pruebas unitarias en entornos Mainframes y análisis del código fuente para realizar un informe sobre la calidad del objeto fuente basándose en métodos empíricos de análisis.<br><br>El resultado de los distintos análisis será trasladado a un documento de diseño técnico del programa. | | |
| **Criterio de aceptación** | Revisión de pruebas integras. | | |
| **Origen e Historia** | Descripción | Fecha | |
| | Añadido | 20-02-2019 | |
| | Aceptada | 05-03-2019 | |

*Tabla 36. Definición requisito REQ-002*

Subrequisito REQ-002-01 definido en tabla 37.

| **ID Requisito:** REQ-002-01 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 20-02-2019 | |
| **Breve descripción** | Generación del código COBOL del programa principal para pruebas. | | |
| **Descripción detallada** | Generar de código en leguaje nativo COBOL para MAINFRAMES, que permita ser utilizado como programa principal para poder realizar las pruebas unitarias del módulo a analizar.<br><br>Este programa principal solamente será generado cuando el programa COBOL a analizar sea un módulo, ya que cuando se trata de programas principales no es necesario ningún programa lanzador, directamente desde el JCL se puede iniciar el programa. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |
| **Origen e Historia** | Descripción | Fecha | |

| | | |
|---|---|---|
| | Añadido | 20-02-2019 |
| | Aceptada | 05-03-2019 |

*Tabla 37. Definición requisito REQ-002-01*

Subrequisito REQ-002-02 definido en tabla 38.

| **ID Requisito:** REQ-002-02 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 22-02-2019 | |
| **Breve descripción** | Generación del entorno de pruebas del lanzador para entornos MAINFRAMES. | | |
| **Descripción detallada** | Partiendo del análisis semántico del programa COBOL, se debe generar un proceso codificado en Job Control Lenguaje (JCL), que permita ejecutar las pruebas unitarias en entornos Mainframes. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 22-02-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 38. Definición requisito REQ-002-01*

Subrequisito REQ-002-03 definido en tabla 39.

| **ID Requisito:** REQ-002-03 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 23-02-2019 | |
| **Breve descripción** | Medición de la calidad y complejidad del código a través de métodos empíricos. | | |
| **Descripción detallada** | Utilización de métodos empíricos para determinar la calidad del programa COBOL y generar un informe con el resultado obtenido. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 23-02-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 39. Definición requisito REQ-002-03*

Subrequisito REQ-002-04 definido en tabla 40.

| **ID Requisito:** REQ-002-04 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 23-02-2019 | |
| **Breve descripción** | Detección automática de fallas o posibles problemas potenciales en el programa a través de muestreos. | | |
| **Descripción detallada** | Mediante un análisis semántico del código fuente, se obtendrá un informe con posibles errores potenciales que puede contener el programa en el momento de su ejecución. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |

| **Origen e Historia** | Descripción | Fecha |
|---|---|---|
| | Añadido | 23-02-2019 |
| | Aceptada | 05-03-2019 |

*Tabla 40. Definición requisito REQ-002-04*

Subrequisito REQ-002-05 definido en tabla 41.

| **ID Requisito:** REQ-002-05 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 25-02-2019 | |
| **Breve descripción** | Crear un grafo con el flujo de caminos que puede tener el programa. | | |
| **Descripción detallada** | Generación automática de un grafo que permita ver de forma gráfica los distintos caminos de ejecución del programa. | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias. | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 25-02-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 41. Definición requisito REQ-002-05*

Subrequisito REQ-002-06 definido en tabla 42.

| **ID Requisito:** REQ-002-06 | | **Tipo:** Requisito funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 25-02-2019 | |
| **Breve descripción** | Generación automática del DT del programa con el formato predeterminado. | | |
| **Descripción detallada** | Obtención de un diseño técnico en formato Word con los siguientes apartados:<br>• PROPÓSITO DEL DOCUMENTO<br>• ARQUITECTURA DE COMPONENTES (HW Y SW)<br>• DIAGRAMA DE EJECUCIÓN<br>• DESCRIPCIÓN DETALLADA DE COMPONENTES<br>• GRAFO DE CAMINOS<br>• INFORME DE CALIDAD POR MÉTODOS EMPÍRICOS<br>• INFORME DE ERRORES POTENCIALES<br>• DOCUMENTOS RELACIONADOS<br>• ANEXOS | | |
| **Criterio de aceptación** | Revisión de pruebas unitarias e íntegras. | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 25-02-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 42. Definición requisito REQ-002-06*

**Requisitos no funcionales en tabla 43.**

| **ID Requisito:** REQ-003 | | **Tipo:** Requisito NO funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |

| | | | |
|---|---|---|---|
| | | 01-03-2019 | |
| **Breve descripción** | Facilidad de uso | | |
| **Descripción detallada** | La utilización de la interface del programa debe ser muy sencilla. | | |
| **Criterio de aceptación** | Revisión de pruebas integras con usuarios no familiarizados con el desarrollo. | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 01-03-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 43. Definición requisito REQ-003*

Requisito REQ-004 no funcional definido en tabla 44.

| **ID Requisito:** REQ-004 | | **Tipo:** Requisito NO funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 01-03-2019 | |
| **Breve descripción** | Disponibilidad de manual de usuario o ayuda al usuario | | |
| **Descripción detallada** | Debe existir un manual de usuario o ayuda para la utilización de la herramienta. | | |
| **Criterio de aceptación** | Revisión de pruebas integras con usuarios no familiarizados con el desarrollo. | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 01-03-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 44. Definición requisito REQ-004*

Requisito REQ-005 no funcional definido en tabla 45.

| **ID Requisito:** REQ-005 | | **Tipo:** Requisito NO funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 02-03-2019 | |
| **Breve descripción** | Rendimiento óptimo | | |
| **Descripción detallada** | Control del rendimiento de los procesos | | |
| **Criterio de aceptación** | Cuantificar el tiempo de los procesos | | |
| **Origen e Historia** | Descripción | | Fecha |
| | Añadido | | 02-03-2019 |
| | Aceptada | | 05-03-2019 |

*Tabla 45. Definición requisito REQ-005*

Requisito REQ-006 no funcional definido en tabla 46.

| **ID Requisito:** REQ-006 | | **Tipo:** Requisito NO funcional | |
|---|---|---|---|
| **Estado** | Nuevo | **Fecha de alta** | **Última revisión** |
| | | 02-03-2019 | |
| **Descripción** | Seguimiento del estado del proceso | | |
| **Descripción detallada** | Ofrecer al usuario el estado de los procesos. | | |

| | | |
|---|---|---|
| **Aceptación** | Satisfacción de usuarios . | |
| **Origen e Historia** | Descripción | Fecha |
| | Añadido | 02-03-2019 |
| | Aceptada | 05-03-2019 |

*Tabla 46. Definición requisito REQ-006*

## *3.5 Descripción de la solución*

El objeto de este apartado es describir la traducción de los requisitos definidos en el apartado 5.2 en una especificación de diseño técnico que satisfaga estos requisitos, debiendo existir una trazabilidad entre éstos y la solución aportada.

**Diagrama de flujo de datos (DFD), en figura 26.**

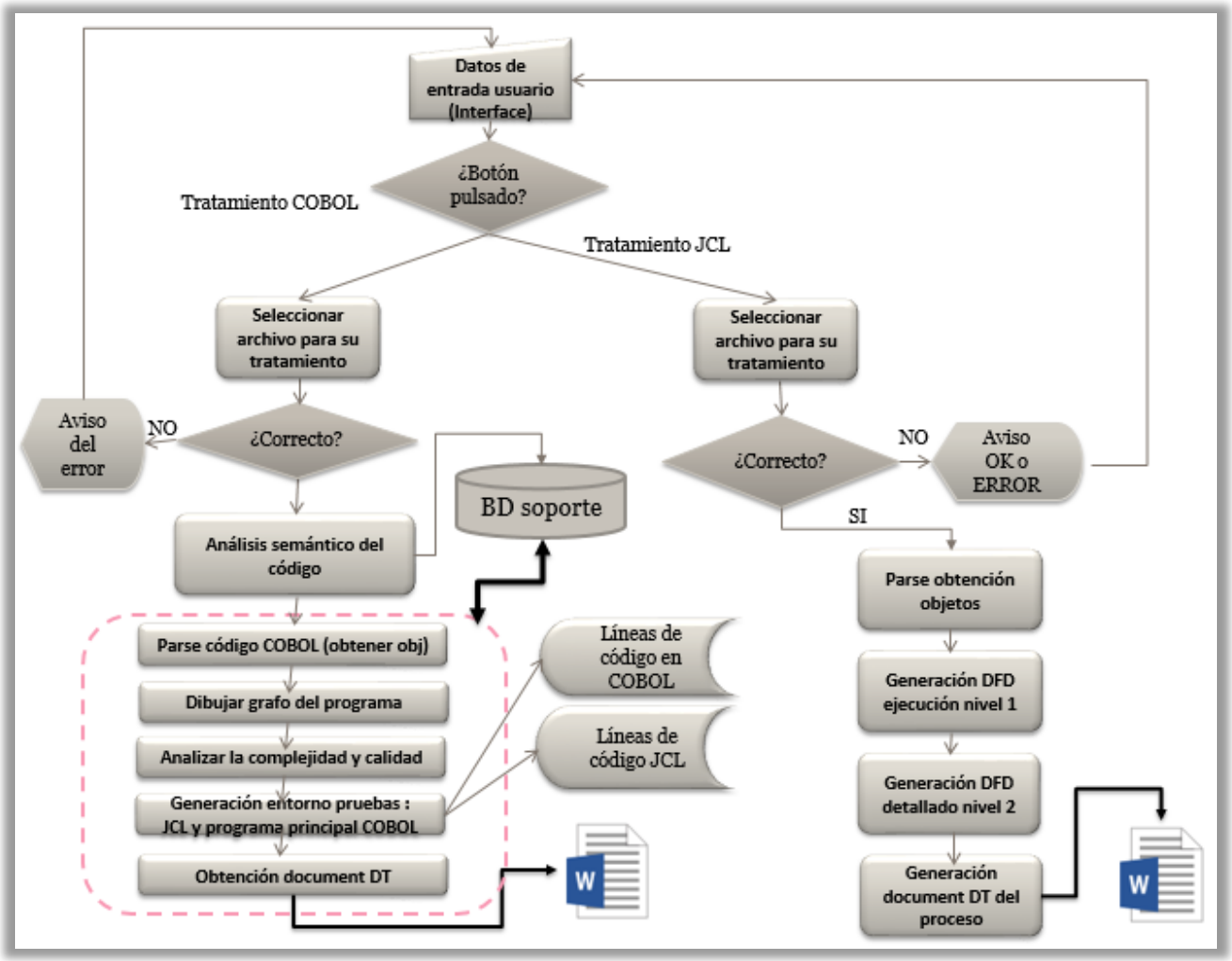



*Figura 26. Diagrama de flujo de datos*

**Arquitecturas planteadas**

En la figura 27 se plantea la arquitectura para el requisito REQ-001 Generación JCL.

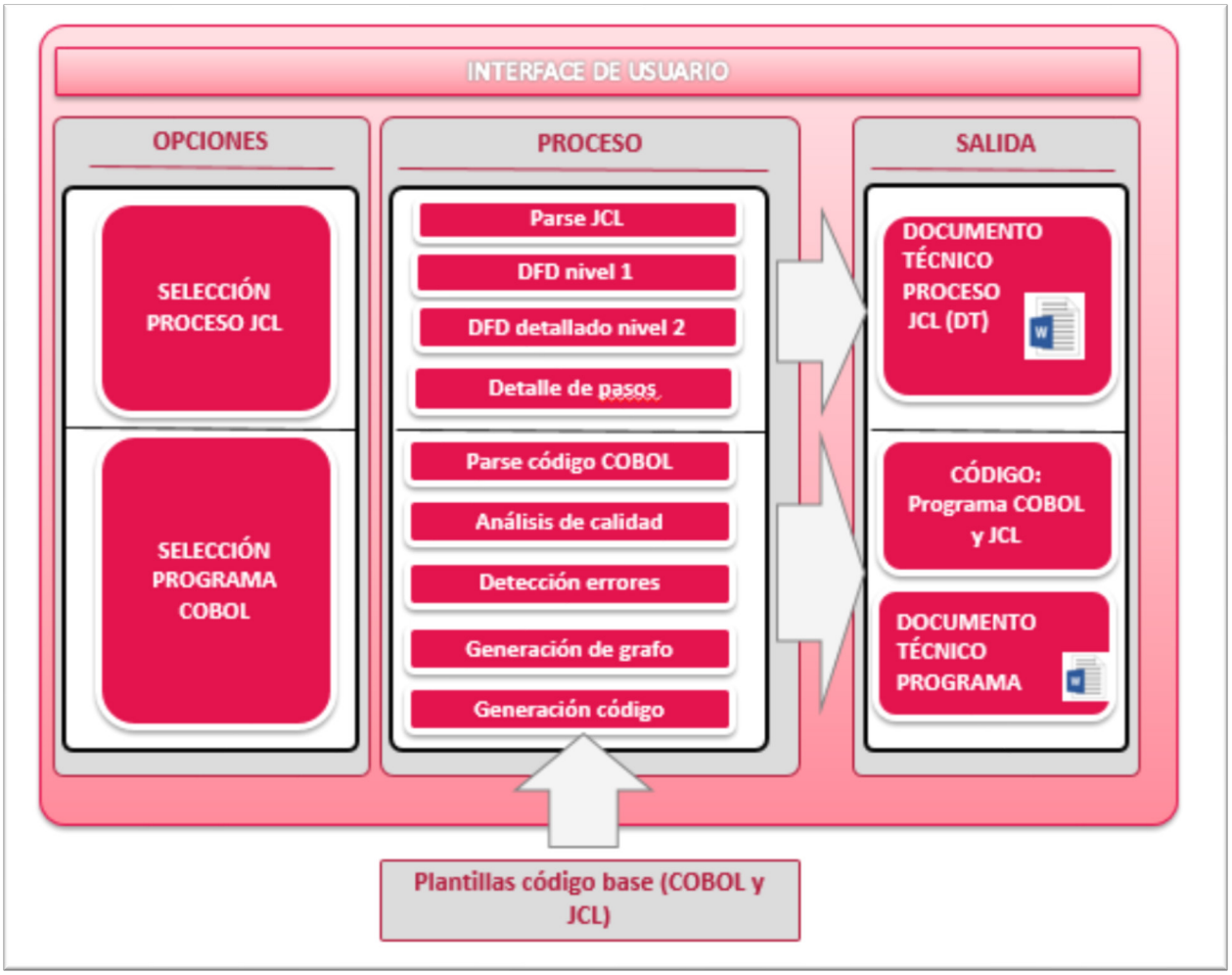



*Figura 27. Arquitectura planteada solución global*

**Elementos de software utilizados y desarrollados.**

El lenguaje de programación que va a ser utilizado es el VB y la plataforma para su desarrollo será la hoja de cálculo Excel.

La idea es disponer de una herramienta de desarrollo sencilla que ya disponga de una interface de usuario que se pueda utilizar fácilmente.

Para la realización de las pruebas, se utilizarán las siguientes herramientas:

- Generación en VB. La herramienta de desarrollo de VB de Excel.
- Generación COBOL. Emulación 3270 para arquitecturas Z.
- Generación JCL. Emulación 3270 para arquitecturas Z.

### 3.3.1 Planteamiento de la solución REQ-001

El planteamiento de la solución del requerimiento REQ-001 (Generación automática de documentación técnica sobre procesos JCL) estará basado en un enfoque MDA, tal como se comenta en el apartado 1.4 Arquitectura planteada y se describe en la figura 9 ( Arquitectura en 3 niveles MDA para GAD) y usando modelo de dominio (un modelo de dominio es un conjunto de abstracciones que facilitan la transformación de los conceptos y comportamientos del mundo real).

en conocimiento y parámetros de software que permitan la resolución de determinados problemas.

**Definición del metalenguaje REQ-001**

En este apartado se va a definir la síntesis del metalenguaje que será utilizado para realizar la transformación al documento final a partir del código fuente seleccionado.

Para poder abarcar las características y dominios necesarios para la transformación, se utilizará la siguiente interfaz en forma de tabla, relacionada en la tabla 47.

| Job | Paso | Num Paso | Programa | Fic | DSN | E | S | RECFM | LRECL | PARM | DISP | COMENTARIOS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AG1SORT | PA9999 | 1 | SORT | SORTJNF1 | TSDE.FX.AN.FXJXML03.XMLM290 | V | F | | 0 | | | Ordenación fichero |
| AG1SORT | PA9999 | 1 | SORT | SORTJNF2 | TSDE.AG1.PRUE | V | F | | 0 | | | |
| AG1SORT | PA9999 | 1 | SORT | SORTJNF3 | DUMMY | V | F | | 0 | | | |
| AG1SORT | PA9999 | 1 | SORT | BOTH | TSDE.CRUCE.CONT.PERCONT | F | V | FB | 60 | | | |
| AG1SORT | PA9999 | 1 | SORT | F1ONLY | TSDE.CRUCE.CONT.PERCONT | F | V | FB | 50 | | | |
| AG1SORT | PA9999 | 1 | SORT | F2ONLY | TSDE.CRUCE.CONT.PERCONT | F | V | FB | 35 | | | |
| | | | | | | | | | | | | |

*Tabla 47. Definición del metalenguaje*

En la tabla 48 se describen las columas.

| Campos | Descripción | Tipo |
|---|---|---|
| **Job** | Nombre del job del JCL | Alfanumérico |
| **Paso** | Nombre del paso | Alfanumérico |
| **NumPaso** | Posición del paso dentro del JCL | Numérico |
| **Programa** | Nombre del programa que se ejecuta dentro del paso. Podrá hacer referencia a un programa COBOL o a una herramienta predefinida | Alfanumérico |
| **Fic** | Nombre corto del fichero | Alfanumérico |
| **DSN** | Puntero externo de referencia para localizar el fichero | Alfanumérico |
| **Entrada** | Indicador si es un fichero de entrada | Booleano |
| **Salida** | Indicador si es un fichero de salida | Booleano |
| **RECFM** | Tipología del fichero (solamente salida). Valores:<br>• FB. Fijo bloqueado.<br>• VB. Fichero de longitud variable | Alfanumérico |
| **LRECL** | Longitud del fichero | Numérico |
| **PARM** | Parámetros de entrada en el paso | Alfanumérico |
| **DISP** | Disposición del fichero de salida.:<br>(VALOR1, VALOR2, VALOR3)<br>• VALOR1. Los valores válidos son **NEW** para fichero nuevos que no existen y se crearán u **OLD** para ficheros ya creados.<br>• VALOR2. Se indica **CATLG** para crear el fichero si el proceso finaliza con retorno 0 y **UNCATLG** para no catalogarlo si finaliza con retorno 0.<br>• VALOR3. Se indica **CATLG** para crear el fichero si el proceso finaliza con retorno "DIFERENTE DE" 0 y **UNCATLG** para no catalogarlo si finaliza con retorno diferente de 0. | Alfanumérico |
| **COMENTARIOS** | Breve comentario sobre la funcionalidad del paso | Alfanumérico |

*Tabla 48. Descripción campos metaleguaje*

**Estructuras utilizadas en REQ-001.**

En los siguientes apartados se definirán los formatos de las estructuras de salida.

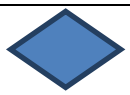
Estructuras DFD.

El DFD (Diagrama de Flujo de Datos), es una representación visual del funcionamiento global de una cadena, representando todos sus elementos y sus interfaces de E/S con elementos gráficos cómo rectángulos, círculos, flechas, etc.

El DFD es ampliamente utilizado para los desarrollos estructurados y se pueden realizar 3 grandes particiones por niveles:

- **Nivel 0**: Diagrama de contexto. Alto nivel, para ver las relaciones entre sistemas o aplicaciones.
- **Nivel 1**: Diagrama de nivel superior. Nivel utilizado para describir el flujo de datos en aplicaciones o cadenas, describiendo las entradas y salidas.
- **Nivel 2**: Diagrama de detalle o expansión. Detalle de flujo de datos a nivel de elemento de Software, describiendo los caminos alternativos que puede tomar los datos.

Los elementos más importantes del DFD se enumeran en la tabla 49.

| Elemento | Forma | Descripción |
|---|---|---|
| Inicio/<br><br>Terminador |  | Rectángulo con las esquinas curvas. Colocado en el inicio y en el final del DFD para indicar el inicio del proceso y el final. |
| Procesos |  | Rectángulo, indicando el nombre del proceso en su interior. Hace referencia a una acción en concreto. Para nuestra representación, harán referencias al paso de la cadena y al nombre del programa |
| Subprocesos |  | Representado con un rectángulo y líneas verticales en los laterales del rectángulo. Indica de subprocesos predefinidos, como llamadas a módulos. (no utilizado) |
| Decisión |  | Rombo. Utilizado para tomar un camino entre n dependiendo de una determinada condición |
| Conector |  | Círculo. Se utiliza para realizar conexiones entre |

| | | |
|---|---|---|
| | | páginas o conexiones entre diferentes partes del DFD. En nuestro caso se utilizará para conectar y hacer referencia a ficheros que se utilizan en la cadena y que no se encuentran en paso consecutivos. |
| Fechas |  | Representación de una flecha. Indica la dirección de ejecución del proceso (el flujo que toma la información) |
| Datos almacenados |  | Para la representación de los ficheros de E/S |
| Documentos<br><br>Impresión |  | Indicará que los datos son obtenidos de una tabla DB2 |
| Comentarios |  | Inclusión de comentarios |

*Tabla 49. Elementos gráficos DFD*

Elementos JCL

En la tabla 50 se enumeran los componentes más comunes que contiene un JCL y que nos serán de utilidad para poder extraerlos en la generación de parse.

| Sentencia | Descripción |
|---|---|
| **Comentario** | Incluir comentarios dentro de la cadena de Jcl. En las posiciones 1, 2 y 3 debe informarse de //* y a continuación el comentario que se desee exponer. |
| **Sentencia JOB** | Es el indicador del inicio de JOB. La sentencia JOB consiste en los caracteres // situados en las columnas 1 y 2<br>y cuatro campos: nombre, operación (JOB), parámetros y comentarios.<br>Se requiere una sentencia JOB por cada trabajo. |
| **Sentencia PROC** | Indica los distintos procedimientos que tiene la cadena. Estos procesos se corresponden a programas predefinidos por el SO como por ejemplo COPIAS, |

| | |
|---|---|
| | SORT, DESCARGAS, ETC o con programas definidos por los desarrolladores. Un JOB puede tener n procedimientos. |
| **Sentencia EXEC** | Identifcación del progma que se va a ejecutar. |
| **Sentencia DD** | Definición de datos. Utilizada para identificar y configurar las interfases de E/S a los programas. |
| **DD especiales** | Definición de datos que son especiales |
| **Delimitadoras** | Utilizada para delimitar el final de los datos o registros. |

*Tabla 50. Sentencias JCL*

Relación entre elementos JCL y DFD nivel 1.

En este nivel se obtendrá una visión global de la ejecución del proceso. En este nivel 1, solamente se utilizarán los elementos de Inicio, flechas, Procesos o subproceso y final.

Para facilitar la labor de conversión de texto a gráfico, se definirá una relación entre ambos sistemas.

Para definir esta correspondencia, se definirá un modelo relacional entre elementos y un flujo de ejecución que permita la conversión. Si es necesario, también se definirá otro modelo intermedio que sirva de facilitador.

En la tabla 51 se define el modelo propuesto para la conversión en este nivel.

| Sentencia JCL (inicial) | Paso intermedio | Elemento DFD (final) |
|---|---|---|
| JOB | Detectar sentencia JOB y el nombre de la cadena. Solo puede aparecer una vez. | Inicio |
| EXEC | En la misma línea de este comando, se encuentra el nombre del proceso, el cual debe ser utilizado para incluir dentro de la figura correspondiente en el DFD. Pueden existir n elementos de este tipo dentro de una cadena. | Procesos |
| EXEC de sistemas | Son subprocesos definidos dentro del sistema: COPI, SORT, LOAD, UNLOAD. Pueden existir n. | Subprocesos |
| IF | Sentencia condicional. Es una | Decisión |

| | | |
|---|---|---|
| | sentencia poco habitual. | |
| | Flujo de la ejecución. | Fechas |
| Final de la cadena | Al llegar al final de lectura de la cadena | Final |

*Tabla 51. Relación elementos JCL y DFD nivel 1*

Relación entre elementos JCL y DFD nivel 2.

El Nivel 2. DFD más detallado con las entradas y salidas en cada uno de los procesos o subprocesos. En la gráfica generada para este nivel, se utilizarán el resto de elementos.

En la tabla 52 se define el modelo propuesto para el nivel 2.

| Sentencia JCL (inicial) | Paso intermedio | Elemento DFD (final) |
|---|---|---|
| JOB | Detectar sentencia EXEC y el nombre de la cadena. Solo puede aparecer una vez. | Inicio |
| EXEC | En la misma línea de este comando, se encuentra el nombre del proceso, el cual debe ser utilizado para incluir dentro de la figura correspondiente en el DFD. Pueden existir n elementos de este tipo dentro de una cadena. | Procesos |
| EXEC de sistemas | Son subprocesos definidos dentro del sistema: COPI, SORT, LOAD, UNLOAD. Pueden existir n. | Subprocesos |
| IF | Sentencia condicional. Es una sentencia poco habitual. | Decisión |
| DSN no consecutivos en E/S | Los ficheros de salida que no se utilicen en el paso consecutivo, se deberán referenciar con conectores. | Conector |
| | Flujo de la ejecución. | Fechas |
| Rectángulo con los laterales curvados en forma de C. Nos indicará los ficheros, que dependiendo de la dirección de la flecha hacia el proceso, indicará si es de entrada o de salida. | | Datos almacenados |
| Indicará que los datos son obtenidos de una | | Tabla de BD |

| | | |
|---|---|---|
| tabla DB2 | | |

*Tabla 52. Relación elementos JCL y DFD nivel 2*

Estructura para el documento técnico DT

El documento técnico, será un archivo tipo Word con la siguiente estructura:

**Índice**

**HOJA DE CONTROL**

**1 PROPÓSITO DEL DOCUMENTO**

**2 ARQUITECTURA DE COMPONENTES (HW Y SW)**

**3 DIAGRAMA DE EJECUCIÓN (DFD NIVEL 1)**

**4 DIAGRAMA DETALLADO (DFD NIVEL 2)**

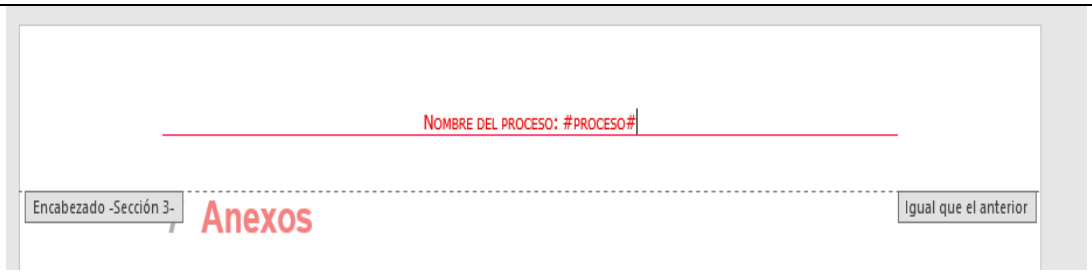
**5 DESCRIPCIÓN DETALLADA DE COMPONENTES**

**6 DOCUMENTOS RELACIONADOS**

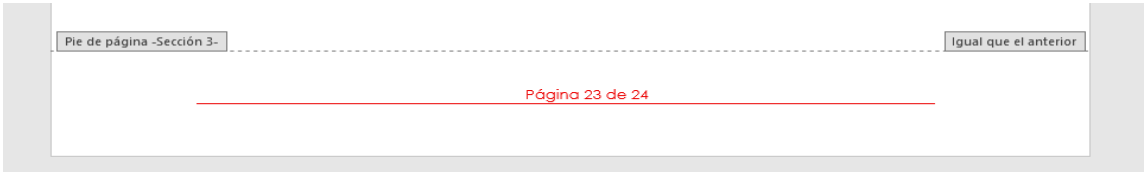
**ANEXOS**

Los formatos del documento se definen en la tabla 53.

| | **FORMATOS** |
|---|---|
| **Formato títulos:** Arial 18 color rojo<br>**Formato texto:** Arial 12 color gris claro.<br>**Cabecera:** Tahoma 9 normal color rojo. Con el siguiente formato: | |




**Pie**: Century 9 normal color rojo, con formato:




## Hoja de control

**El formato será el siguiente:**

Hoja de control del documento

**Control de la Plantilla**

*Información a editar únicamente cuando se realice un cambio de la plantilla.*

| Versión | Fecha de publicación | Descripción de los cambios |
|---|---|---|
| 1.0 | 19/04/2019 | N/A |

**Control del Entregable**

*Información a editar por el responsable del entregable.*

**Responsable:**

Abel García Sánchez

**Destinatarios:**

- Laboratorio de desarrollo

| Versión | Fecha (*) | Descripción de los cambios |
|---|---|---|
| 1.0 | #fecha# | Versión Inicial |
| | | |

* NOTA 1: *La fecha de la portada habrá de coincidir con la fecha de la última versión incluida en la tabla.*

*NOTA 2: El texto en cursiva del presente documento representa instrucciones para completar cada capítulo por lo que debe sustituirse por el contenido que corresponda en cada caso.*

*La letra con la que se complete cada capítulo ya no será cursiva.*

| Apartado 1 | PROPÓSITO DEL DOCUMENTO |
|---|---|

**Insertar el siguiente literal:**

El presente documento tiene por objeto describir la traducción del Análisis funcional al planteamiento técnico y a una especificación de diseño para la solución informática correspondiente al proceso: ***<nombre del proceso>***

| Apartado 2 | ARQUITECTURA DE COMPONENTES (HW Y SW) |
|---|---|

**Insertar el siguiente literal y tabla:**

A continuación, se enumeran los componentes afectados en la solución de enfoque táctico.

Todo el desarrollo se realiza sobre la instalación existente, sin cambios en la arquitectura base.

| ELEMENTO | TIPO | PASO UBICADO | DESCRIPCIÓN |
|---|---|---|---|
| #elem1# | #tipo1# | #paso1# | #descrip1# |
| #elem2# | #tipo2# | #paso2# | #descrip2# |
| #elem3# | #tipo3# | #paso3# | #descrip3# |
| #elem4# | #tipo4# | #paso4# | #descrip4# |

| Apartado 3 | DIAGRAMA DE EJECUCIÓN (DFD NIVEL 1) |
|---|---|

**Insertar el siguiente literal y tabla:**

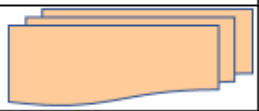
En el siguiente diagrama se muestra el flujo de información y de ejecución de la información del proceso : **#proceso#.**

1 <tabla para insertar DFD nivel 1>

| Apartado 4 | DIAGRAMA DE EJECUCIÓN (DFD NIVEL 2) |
|---|---|

**Insertar el siguiente literal y tablas:**

En la siguiente tabla se muestra los objetos utilizados en la generación del DFD detallado:

| Dibujo | Descripción | Dibujo | Descripción |
|---|---|---|---|
|  | Ficheros de E/S generados en el proceso |  | Paso ejecutado (programa o aplicación predeterminada |
|  | Entradas externas al proceso |  | Fichas SYSIN (utilizadas en SORT) |
|  | Parámetros utilizados (condiciones, etc) |  | Listados generados |
|  | Flechas de conexión |  | Flechas para indicación de comentarios |
|  | Salida spool |  | Ficheros DUMMY (vacíos) |
|  | Comentarios | | |

A continuación, se muestra el diagrama de flujo de datos en detalle:

2 <insertar DFD detallado de nivel 2>

| Apartado 5 | DESCRIPCIÓN DETALLADA DE COMPONENTES |
|---|---|

**Insertar 22 tablas con los siguientes literales y variables para sustituir:**

**Nombre del paso: #nompaso1#.**

**Identificación:**

#nomproceso1#

**Tipo:**

#tipoproceso1#

**Nuevo/Modificado/Sin cambios:**

Sin cambios

**Ficheros de entrada:**

#ficherosentrada1#

**Ficheros de salida:**

#ficherossalida1#

**Comentarios:**

#datoscomentarios1#

**Nombre del paso: #nompaso2#.**

**Identificación:**

#nomproceso2#

| Apartado 6 | DOCUMENTOS RELACIONADOS |
|---|---|

**Insertar los siguientes literales y tablas:**

*Se incluirá en este apartado toda la documentación a la que se haga referencia a lo largo de este documento*

*La utilización de este capítulo es OPCIONAL.*

| Ref. | Código | Nombre |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

| Apartado 7 | ANEXOS |
|---|---|
| Espacio reservado para la inclusión de anexos. | |

*Tabla 53. Formato documento DT proceso JCL*

**PSEUDOCÓDIGO del ModuloGenerarDFD.**

Párrafo principal

<Inicializar_Datos> 'Iniciar las variables utilizadas en el proceso

<Obtener_elementos_JCL> 'Parse del proceso JCL (análisis semántico)

<Generar_DFD_Nivel1>

<Ggenerar_DFD_Nivel2>

< Visualizar_objetos_JCL>

< Generar_documento_DT_proceso>

Obtener_elementos_JCL

<abrir fichero> 'abrir fichero seleccionado con el código JCL

Mientras NO EOF fichero

<Leer_sentencia>

<Buscar_ficha_extendida>

< Buscar_etiquetas>

< Buscar_librerias_ficheros>

'registrar en la estructura del metalenguaje definido

Fin-Mientras

Cerrar ficheros I/O.

generar_DFD_Nivel1

```
<borrar_hoja_dfdNivel1> ‘borramos hoja, por si existía de otro proceso

<crear_hoja_dfdNivel1> ‘se crea una nueva: DFD_N1_ + nombre del JOB

<inicializar_datos_dfd1>

<dibujar inicio>

Desde 1 hasta último elemento estructura metamodelo

    Si objeto = paso

        <dibujar objeto>

        <enlazar_siguiente_objeto>

    Fin si

    Add 1 elemento

Fin desde

<dibujar terminador>
```

generar_DFD_Nivel2

```
<borrar_hoja_dfdNivel2> ‘borramos hoja, por si existía de otro proceso
<crear_hoja_dfdNivel2> ‘se crea una nueva: DFD_N2_ + nombre del JOB
<inicializar_datos_dfd2>
<preparar_hoja> ‘formatear la hoja de cálculo para inicial la generación
<dibujar inicio>
Desde 1 hasta último elemento estructura metamodelo
    Mientras mismo paso
      <detectar_forma_y_dibujar>
      <incluir_parametros>
      <conectar_forma>
      <insertar_comentarios>
      <detectar_ficheros_y_dibujar> ‘detectar si son de entrada para conectar con
                                    ‘predecesor
       <detectar_fichas_dibujar>
          <dibujar objeto>
          <enlazar_siguiente_objeto>
    Fin mientas
Fin desde
```

```
visualizar_objetos_JCL ‘se visualizan en una nueva hoja para comprobar que están OK
```

```
<borrar_hoja_OBJETOS_JCL> ‘borramos hoja, por si existía de otro proceso
<crear_hoja_OBJETOS_JCL>
Desde 1 hasta último elemento estructura metamodelo
    <volcar_a_hoja> ‘volcamos la información del elemento a la hoja Excel
                    ‘con el mismo formato del metamodelo, para su revisión
    Add 1 elemento
Fin desde
```

```
Generar_documento_DT_proceso
```

```
<inicializar_objeto_Word>
<seleccionar_plantilla >
<sustituir variables >  ‘sustitución de todas las variables de finidas en la plantilla
                ‘por los datos del metamodelo (programas, pasos, ficheros, etc)
<insertar imagen DFD nivel1>
<insertar imagen DFD nivel2>
<limpiar hojas y espacios vacíos> ‘se realiza una limpieza de espacios que han
            ‘ quedado sin rellenar en el documento
<guardar documento> ‘salvar el documento con el nombre
                        SALIDA_DT_<nombre proceso>_1.0.doc
```

### 3.3.2 Planteamiento de la solución REQ-002

El planteamiento de la solución del requerimiento REQ-002 (Generación automática de entornos de prueba Mainframes y analizador de código COBOL).

Para esta solución se utilizará un enfoque MDA para definir una arquitectura de alto nivel y sub-enfoques para cada uno de los subrequisitos, para intentar disminuir la complejidad del desarrollo.

En la figura 28 se puede ver el diseño de la arquitectura de alto nivel del requisito basada en MDA, con planteamientos mixtos en sub-requisitos:

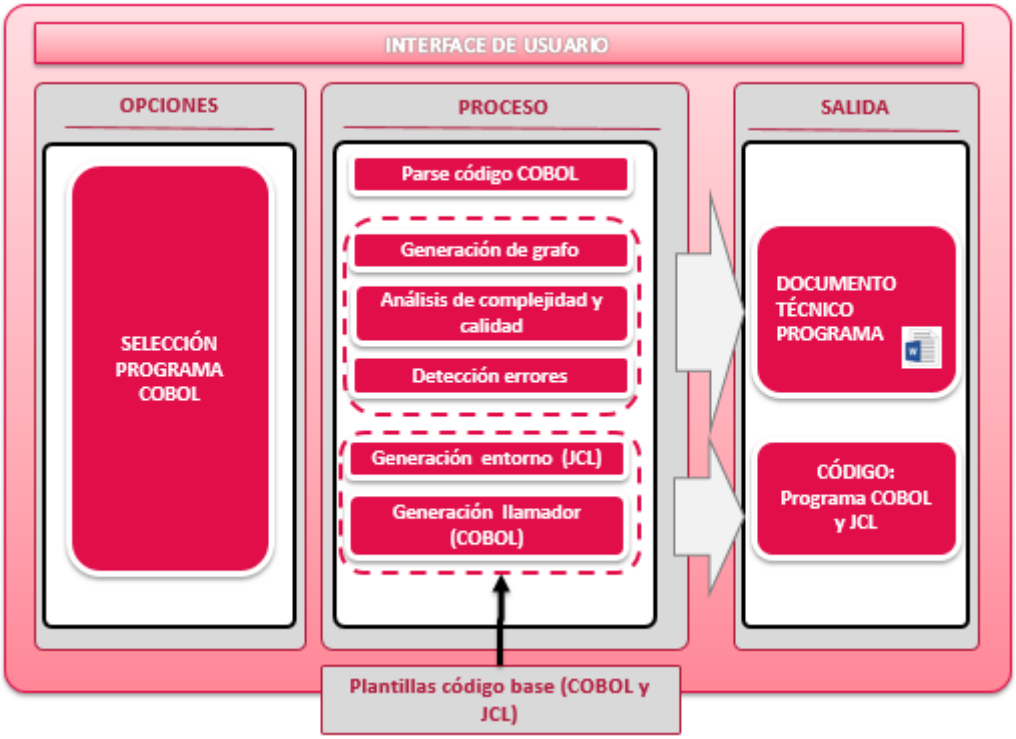



*Figura 28. Arquitectura en 3 niveles MDA*

**Relación entre subrequisitos y funcionalidades.**

En la tabla 54 se relacionan los requisitos y las funcionalidades

| Requisito | Descripción Requisito | Relación funcionalidad |
|---|---|---|
| REQ-002-01 | Generación del código COBOL del programa principal para pruebas. | Generación llamador (COBOL) |
| REQ-002-02 | Generación del entorno de pruebas del lanzador para entornos MAINFRAMES (generación del JCL).. | Generación entorno (JCL) |
| REQ-002-03 | Medición de la calidad y complejidad del código a través de métodos empíricos | Análisis complejidad y calidad |
| REQ-002-04 | Detección automática de fallas o | Detección de errores |

| | | |
|---|---|---|
| | posibles problemas potenciales en el programa a través de muestreos | |
| REQ-002-05 | Crear un grafo con el flujo de caminos que puede tener el programa. | Generación de grafo |
| REQ-002-06 | Generación automática del DT del programa con el formato predeterminado. | Salida documento DT |
| TODOS | Todos los subrequisitos | Parse código COBOL |

*Tabla 54. Relación entre requisitos y funcionalidades*

**Definición del metalenguaje y estructuras para REQ-002.**

Se definirán dos estructuras de datos para poder dar solución a las distintas funcionalidades de los subrequisitos. A continuación, se describen:

Estructura para registrar los **objetos cobol** y ejemplo de valores en la tabla 55.

| Tipo | Nombre | Definición | Longitud | Observaciones | Linkage | Nivel |
|---|---|---|---|---|---|---|
| Nombre del programa | FXP19318 | numerico | 5 | Warning. No utilizado | FALSO | |
| VARIABLE | VARIABLE1 | alfanumerico | 6 | OK. Variable utilizada | FALSO | 1 |
| VARIABLE | VARIABLE2 | alfanumerico | 2 | Warning. No utilizado | FALSO | |
| ALTERNATIVA | IF | Condición: A > B | 0 | OK. Condición cerrada | FALSO | |
| ALTERNATIVA | IF | Condición: C > B | 0 | OK. Condición cerrada | FALSO | |
| MODULO | MODULO1 | numerico | 5 | Warning. No utilizado | FALSO | |

*Tabla 55. Estructura objetos COBOL*

Descripción de las columnas del metamodelo de objetos COBOL, en la tabla 56.

| Campos | Descripción | Tipo |
|---|---|---|
| Tipo | Tipo de objeto COBOL. Algunos de los posibles valores son los siguientes:<br>• Nombre del programa. Descripción implícita en nombre.<br>• VARIABLES. Variables utilizadas.<br>• ALTERNATIVAS. Instrucciones condicionantes.<br>• MODULOS. Programas o rutinas a las que son llamadas desde el programa.<br>• ETC. | Alfanumérico |
| Nombre | Nombre. Nombre del objeto. | Alfanumérico |
| Definición | Si el tipo es una VARIABLES, indica la tipología, es decir si es numérica, alfanumérica, etc.<br>Si es una ALTARNATIVA, se incluye la condición.<br>Para otras tipologías sin información | Numérico |
| Longitud | Solamente utilizado para VARIABLES. Indica la longitud de la variable. | Numérico |
| Observaciones | Análisis del objeto. Si contiene errores leves o graves, se indicarán en este campo. | Alfanumérico |
| Linkage | Para indicar si el objeto es de intercambio | Booleano |
| Nivel | Solamente utilizado para VARIABLES. Se utilizará para indicar estructuras principales del programa | Numérico. |

*Tabla 56. Descripción de campos estructura objetos Cobol*

Estructura para registrar las **líneas del código cobol en tabla 57.**

| Entrada | Sentencia | Loc proceso | Localización | Variable1 | Variable2 | Variable3 | Variable4 | Camino asignado |
|---|---|---|---|---|---|---|---|---|
| 1 | IDENTIFICATION | | IDENTIFICATION | | | | | 1 |
| 2 | PROGRAM-ID | | IDENTIFICATION | FXP19318 | | | | 1 |
| ... | | | | | | | | |
| 16 | 77 | | WORKING-STORAGE | | | | | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 17 | PROCEDURE | | PROCEDURE | | | | | 1 |
| 18 | INICIO | | PROCEDURE | | | | | 1 |
| 19 | IF | | PROCEDURE | A | > | B | | 1 |
| 20 | DISPLAY | | PROCEDURE | "A > B" | | | | 2 |
| 21 | MOVE | | PROCEDURE | A | TO | B | | 2 |
| 22 | END-IF | | PROCEDURE | | | | | 2 |

*Tabla 57. Estructura registro objetos COBOL*

Descripción de las columnas en tabla 58.

| Campos | Descripción | Tipo |
|---|---|---|
| **Entrada** | Número secuencial de entrada a la tabla interna e índice la misma | Numérico |
| **Sentencia** | Sentencia del código COBOL que ha sido detectada | Alfanumérico |
| **Loc proceso** | Si la sentencia se encontraba dentro de un proceso, éste se indica en este campo. | Numérico |
| **Localización** | Localización de la sentencia dentro de la estructura de un programa COBOL, existiendo las siguientes posibilidades:<br>• IDENTIFICATION<br>• ENVIROMENT<br>• FILE<br>• WORKING-STORAGE<br>• PROCEDURE | Alfanumérico |
| **Variable1,2,3,4** | Variables o literales utilizadas en la instrucción | Alfanumérico |
| **Camino asignado** | Camino asignado a la instrucción | Alfanumérico |

*Tabla 58. Descripción estructura líneas de código*

**Funcionalidad Parse código cobol.**

Obtención de la información definida en los metamodelos a través de análisis semántico.

En esta funcionalidad, se utilizará una metodología de generación de código parametrizada, donde se utilizarán unas tablas paramétricas de una SGBD Access en la cual se definirán los criterios de extracción y conversión del código COBOL a los metamodelos definidos en los apartados anteriores.

El método consiste en detectar las sentencias del programa origen y buscar en las tablas paramétricas todas las acciones asociadas en la conversión para dicha sentencia, con las cuales se permita realizar la transformación.

Este sistema permite cierta escalabilidad y adaptación del software para el tratamiento de nuevas sentencias COBOL no incluidas en el planteamiento inicial, permitiendo en muchos adaptarse simplemente con la actualización de las tablas paramétricas, sin la necesidad de modificar el software.

En la figura 29 se muestra la arquitectura planteada:

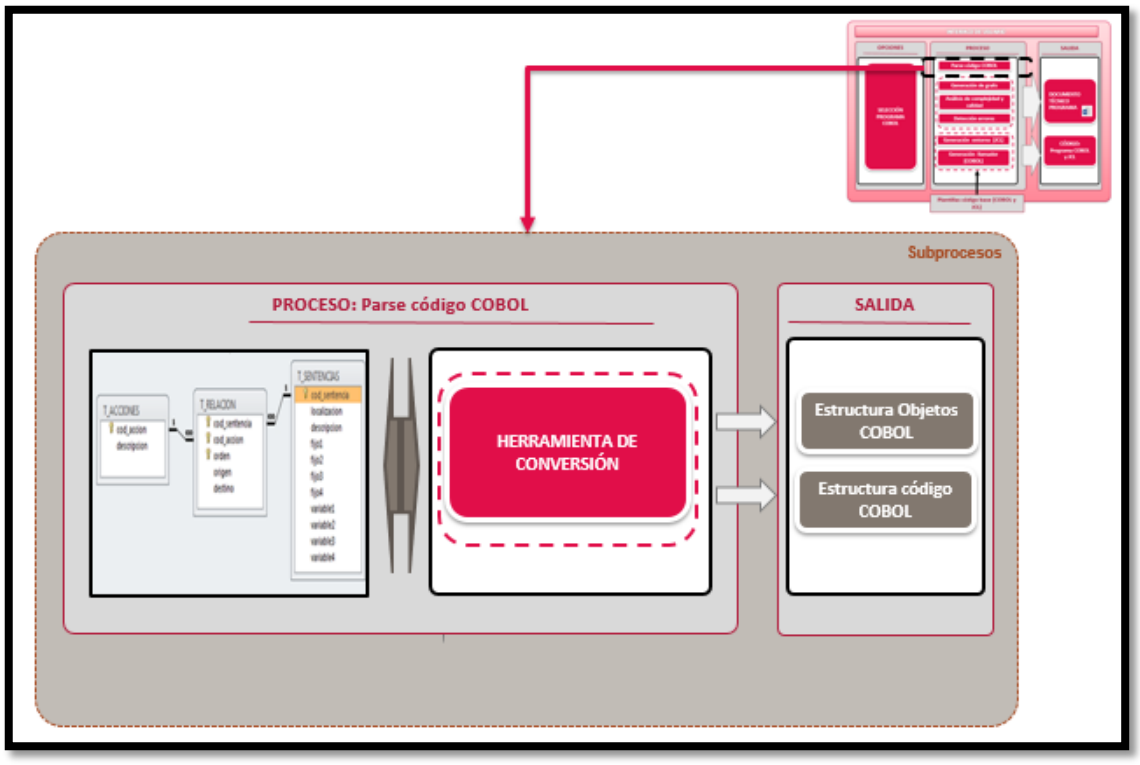



*Figura 29. Arquitectura parse COBOL*

Estructura básica de un programa COBOL.

Para poder realizar los desarrollos, se debe entender y analizar la estructura básica de los programas COBOL. Esta estructura es descrita en la tabla 59.

| DIVISIÓN | Descripción |
|---|---|
| **Identification división** | Definición de identificadores del programa; autor, nombre del programa, etc. |
| **Enviroment division** | Definición de ficheros E/S y de impresoras |
| **File section** | Definición de las características de los ficheros definidos en la FILE-CONTROL |
| **Working–storage section** | Definición de las variables y constantes que serán utilizadas en el programa |
| **Procedure division** | Contiene el algoritmo de ejecución |

*Tabla 59. Estructura básica programa COBOL*

### Identification división

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NombrePrograma.
AUTHOR.  NombreAutor.
```

- Siempre debe ir relleno el campo PROGRAM-ID, con el nombre del programa.
- Siempre debe ir relleno el campo AUTHOR indicando el autor del programa.

### Enviroment division

Ejemplo de esta sección con la entrada de un fichero:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
FILE-CONTROL.
SELECT F-ENTRADA ASSIGN TO ARCHIVO.

 FILE STATUS IS FS-ARCHIVO
```

### File section

```
FD  F-ENTRADA  BLOCK CONTAINS 0 RECORDS.
     LABEL RECORD IS STANDAR.
```

```
        DATA RECORD IS REG-ENTRADA.
01  REG-ENTRADA                PIC X(nnn).
```

Definición de los parámetros de los ficheros incluidos en el apartado de FILE-CONTROL.

**Working–storage section.**

```
WORKING-STORAGE SECTION.
01 VARIABLE1 PIC X(15).
01 VARIABLE2 PIC XX.
01 VARIABLE3 PIC 9(15).
01 VARIABLE4 PIC 9999 VALUE 0.
01 VARIABLE5   PIC    9(15).
01 RESULTADO   PIC    9(15).
01 VARIABLE6   PIC    X.
77 INDICE          PIC 9(5).
01 FECHA.
    05 AAAA PIC 99.
    05 MM PIC 99.
    05 DD PIC 99.
```

Definición de las variables, constantes, y estructuras de tablas.

Los distintos tipos en cobol son los siguientes:

- 9, indica dígito numérico.
- 99v99. Los número a la izquierda de la v indican la parte entera del número y los de la parte derecha los decimales.
- XXXX. Cada X equivale a una carácter alfanumérico.
- X(5). Tanto los 9 cómo las X se pueden definir por repetición o incluyendo entre paréntesis la longitud. En este ejemplo el X(5) equivaldría a una variable alfanumérica de 5 posiciones XXXXX.
- Z. Variables editadas para que no aparezcan los 0 de la izquierda.
- ETC

**Procedure división**

Algoritmo de ejecución del programa.

En esta división se escribirá el código de ejecución con las sentencias secuenciales, alternativas, repetitivas o modulación por sub procesos.

```
PROCEDURE DIVISION.
INICIO.
   DISPLAY "Primer programa"
   DISPLAY "Hola Mundo"
   STOP RUN.
```

**Parametrización.**

Ejemplo de la utilización de la parametrización.

En la tabla 60 se enumaran las sentencias parametrizadas:

| T_SENTENCIAS | | | | |
|---|---|---|---|---|
| **Cod_sentencia** | **Localización** | **Descripción** | **Fijo 1 -4** | **Variable 1-4** |
| 01 | WORKING | VARIABLE | | |
| 05 | WORKING | VARIABLE | | |
| 10 | WORKING | VARIABLE | | |
| 15 | WORKING | VARIABLE | | |
| 77 | WORKING | VARIABLE | | |
| AUTHOR | IDENTIFICATION | Autor del programa | // El autor es: | |
| CALL | PROCEDURE | MODULO | | |
| CLOSE | PROCEDURE | cerrar fichero | | |
| COMPUTE | PROCEDURE | Realizar opciones | | |
| CONFIGURATION | ENVIRONMENT | Clausula sin conversión | | |
| DATA | IDENTIFICATION | REGISTRO | | |
| DISPLAY | PROCEDURE | Visualización | System.out.println( | |
| ELSE | PROCEDURE | Condición alternativa | } else { | |
| END-IF | PROCEDURE | Fin de la condicó | } | |
| END-PERFORM | PROCEDURE | Fin estructura repetitiva | } | |
| ENVIRONMENT | ENVIRONMENT | Clausula sin conversión | | |
| FD | IDENTIFICATION | Relación fichero-registro | | |
| IDENTIFICATION | IDENTIFICATION | Clausula sin conversión | | |
| IF | PROCEDURE | ALTERNATIVA | | |
| LINKAGE | WORKING | Área intercamio | | |
| LINKAJE | WORKING | Área intercamio | | |
| MOVE | PROCEDURE | Mover variables o constantes | = | |

| T_SENTENCIAS | | | | |
|---|---|---|---|---|
| **Cod_sentencia** | **Localización** | **Descripción** | **Fijo 1 -4** | **Variable 1-4** |
| OPEN | PROCEDURE | Apertura de ficheros | | |
| PERFORM | PROCEDURE | Crear sub proceso | () | |
| PERFORM UNTIL | PROCEDURE | BUCLE | while ( | |
| PROCEDURE | PROCEDURE | Clausula sin conversión | | |
| PROGRAM-ID | IDENTIFICATION | Nombre del programa | // Nombre del programa: | |
| SELECT | IDENTIFICATION | FICHERO | | |
| STOP | PROCEDURE | fin del programa | | |
| WORKING-STORAGE | WORKING | Clausula sin conversión | | |

*Tabla 60. Ejemplo datos T_SENTENCIAS*

En la tabla 61 se relacionan las distintas acciones.

| T_ACCIONES | |
|---|---|
| **cod_accion** | **descripcion** |
| activar_linkage | Activamos sw de linkaje para detectar variables intercambio |
| ajustar_caracteres | Ajuste de caracteres de cobol a java |
| anterior_camino | Se regresa al camino anterior |
| asignar_tipo_datos | Asignación del tipo de datos |
| asignar_tipo_fichero | Asignación del tipo de fichero (E/S) |
| buscar_condicion | Busca toda la condición hasta fin de línea |
| buscar_hasta_fin | Recuperar resto de línea |
| buscar_palabra | Leer siguiente palabra y mover |
| cerrar_fichero | se cierra el fichero |
| cerrar_if | Cambiar situación de la sentencia if |
| comprobar_bucle | validar si el bucle es o no correcto |
| comprobar_variable | Verificaciones sobre la variable |
| desactivar_linkage | Activamos sw de linkaje para detectar variables intercambio |
| escribir_salida | Escribir salida |
| guardar_array | Se registra en metamodelo |
| guardar_fichero | guardar nombre del fichero para relacionar con registro |
| guardar_objeto_cobol | guardar datos del objeto cobol |
| guardar_proceso | registro de subprocesos |
| mover_a_fin_linea | Mover hasta el final de la linea |
| mover_blanco | Mover un blanco a salida |

| T_ACCIONES | |
|---|---|
| **cod_accion** | **descripcion** |
| mover_fichero | relación registro con fichero |
| mover_fijas | Mover a salida |
| mover_puntoycoma | Mover punto y como de final sentencia |
| mover_variables | Mover a salida |
| nuevo_camino | Se detecta nuevo camino en algoritmo |
| sin_accion | No realizar ninguna acción |

*Tabla 61. Ejemplo datos T_ACCIONES*

Acciones para el código de sentencia "01".

Se localizará el código de sentencia, que en este caso es 01 y detectará todas las acciones registradas para esta sentencia, ejecutándolas en el orden registrado:

1. Buscar_palabra. Busca la siguiente palabra después de la sentencia y la guarda en la columna 7. En este caso es para localizar el nombre de la variable.
2. Asignar_tipo_datos. Localiza la definición en COBOL
3. guardar_objeto_cobol. Registrar información en la estructura de objetos COBOL.

Sentencia CALL.

Se localizará el código de sentencia, que en este caso es CALL y detectará todas las acciones registradas para esta sentencia, ejecutándolas en el orden registrado:

1. buscar_palabra . Busca la siguiente palabra para localizar el nombre del módulo.
2. guardar_objeto_cobol. Registrar información en la estructura de objetos COBOL.

Las relaciones definidas que hacen posible la conversión del código COBOL a los metamodelos (el análisis semántico), tabla 62.

| T_RELACION | | | | |
|---|---|---|---|---|
| **cod_sentencia** | **cod_accion** | **orden** | **origen** | **destino** |
| 01,05,10 y 77 | asignar_tipo_datos | 2 | | |
| 01,05,10 y 77 | buscar_palabra | 1 | | 7 |
| 01,05,10 y 77 | guardar_objeto_cobol | 3 | 7 | |
| AUTHOR | buscar_palabra | 1 | | 7 |
| AUTHOR | mover_variables | 3 | 7 | |
| CALL | buscar_palabra | 1 | | 7 |

| T_RELACION | | | | |
|---|---|---|---|---|
| cod_sentencia | cod_accion | orden | origen | destino |
| CALL | guardar_objeto_cobol | 2 | 7 | |
| CLOSE | buscar_palabra | 1 | | 7 |
| CLOSE | cerrar_fichero | 3 | | |
| CLOSE | mover_variables | 2 | 7 | |
| COMPUTE | mover_a_fin_linea | 1 | | |
| CONFIGURATION | sin_accion | 1 | | |
| DATA | buscar_palabra | 1 | | 8 |
| DATA | buscar_palabra | 2 | | 7 |
| DATA | guardar_objeto_cobol | 5 | 7 | |
| DATA | mover_fichero | 3 | | |
| DATA | mover_variables | 4 | 7 | |
| DISPLAY | buscar_condicion | 1 | | 7 |
| DISPLAY | mover_variables | 3 | 7 | |
| ELSE | nuevo_camino | 1 | | |
| END-IF | anterior_camino | 1 | | |
| END-IF | cerrar_if | 2 | | |
| END-PERFORM | comprobar_bucle | 1 | | |
| ENVIRONMENT | sin_accion | 1 | | |
| FD | buscar_palabra | 1 | | 7 |
| FD | guardar_fichero | 2 | | |
| FD | mover_variables | 3 | 7 | |
| IDENTIFICATION | sin_accion | 1 | | |
| IF | buscar_palabra | 1 | | 7 |
| IF | buscar_palabra | 2 | | 8 |
| IF | buscar_palabra | 3 | | 9 |
| IF | buscar_palabra | 4 | | 10 |
| IF | comprobar_variable | 8 | | |
| IF | guardar_objeto_cobol | 7 | 7 | |
| IF | mover_variables | 6 | 7 | |
| IF | nuevo_camino | 7 | | |
| LINKAGE | activar_linkage | 1 | | |
| MOVE | buscar_palabra | 1 | | 7 |
| MOVE | buscar_palabra | 2 | | 8 |
| MOVE | buscar_palabra | 3 | | 9 |

| T_RELACION | | | | |
|---|---|---|---|---|
| cod_sentencia | cod_accion | orden | origen | destino |
| MOVE | comprobar_variable | 9 | 7 | |
| MOVE | mover_variables | 4 | 9 | |
| MOVE | mover_variables | 8 | 7 | |
| OPEN | asignar_tipo_fichero | 5 | | |
| OPEN | buscar_palabra | 1 | | 7 |
| OPEN | buscar_palabra | 2 | | 8 |
| OPEN | mover_variables | 3 | 7 | |
| OPEN | mover_variables | 4 | 8 | |
| PERFORM | buscar_palabra | 1 | | 7 |
| PERFORM | guardar_proceso | 3 | 7 | Array_procesos |
| PERFORM | mover_variables | 2 | 7 | |
| PERFORM UNTIL | buscar_palabra | 1 | | 7 |
| PERFORM UNTIL | buscar_palabra | 2 | | 8 |
| PERFORM UNTIL | buscar_palabra | 3 | | 9 |
| PERFORM UNTIL | comprobar_variable | 8 | | |
| PERFORM UNTIL | guardar_objeto_cobol | 6 | 7 | |
| PERFORM UNTIL | mover_variables | 4 | 7 | |
| PERFORM UNTIL | nuevo_camino | 7 | 7 | |
| PROCEDURE | desactivar_linkage | 1 | | |
| PROGRAM-ID | buscar_palabra | 1 | | 7 |
| PROGRAM-ID | guardar_objeto_cobol | 3 | 7 | |
| PROGRAM-ID | mover_variables | 2 | 7 | |
| SELECT | buscar_palabra | 1 | | 7 |
| SELECT | guardar_objeto_cobol | 3 | 7 | |
| SELECT | mover_variables | 2 | 7 | |
| STOP | sin_accion | 1 | | |
| WORKING-STORAGE | sin_accion | 1 | | |

*Tabla 62. Ejemplo datos T_RELACION*

**Funcionalidad: Creación de grafo.**

Esta funcionalidad consiste en obtener los distintos caminos del código analizado y transformarlo a un diagrama tipo grafo con la representación visual de estos caminos.

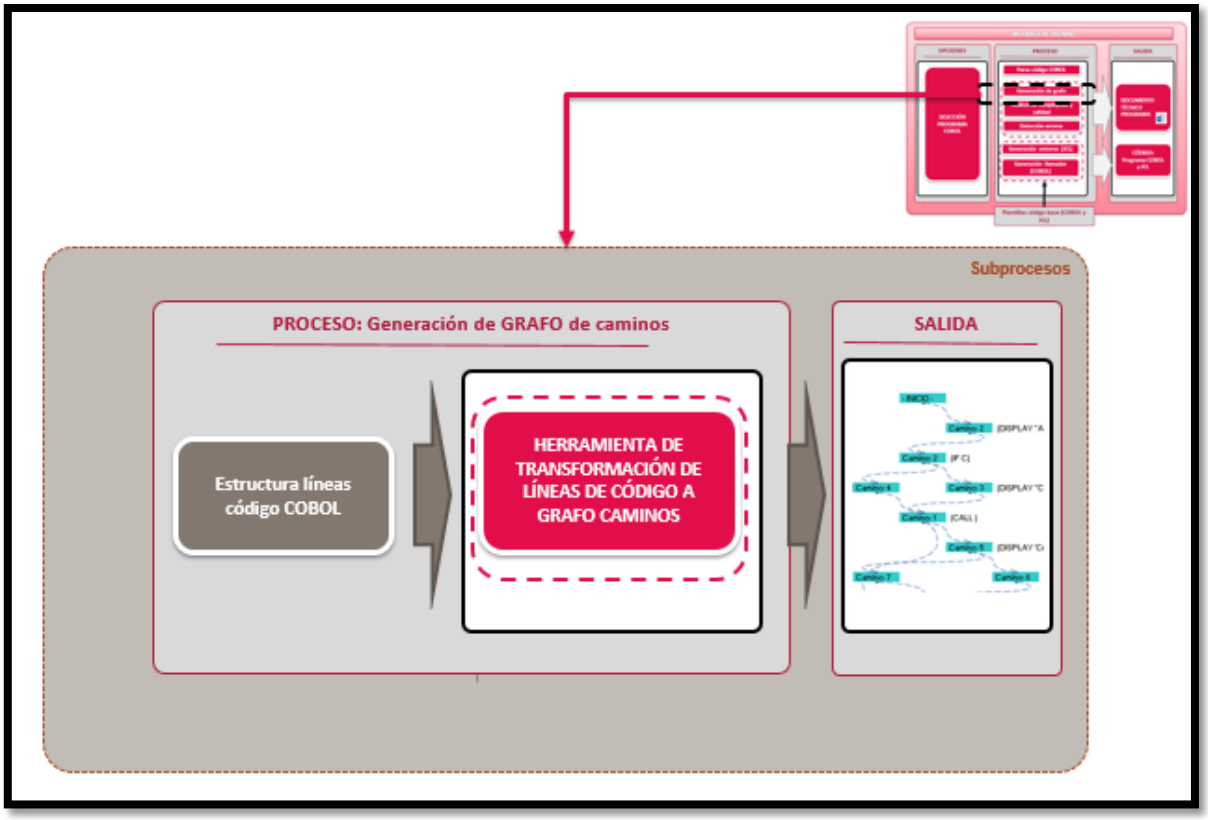



*Figura 30. Arquitectura generación GRAFO*

Tal como se puede observar en la figura 30, se partirá de la estructura de líneas de código COBOL para realizar una transformación a un entorno visual.

**Funcionalidad: Análisis de complejidad y calidad.**

En este análisis de la calidad y complejidad, se utilizarán dos métodos empíricos diferenciados:

- Complejidad CICLOMÁTICA.
- Métricas de calidad HALSTEAD.

En la figura 31 se muestra la arquitectura planteada para la obtención de estas métricas.

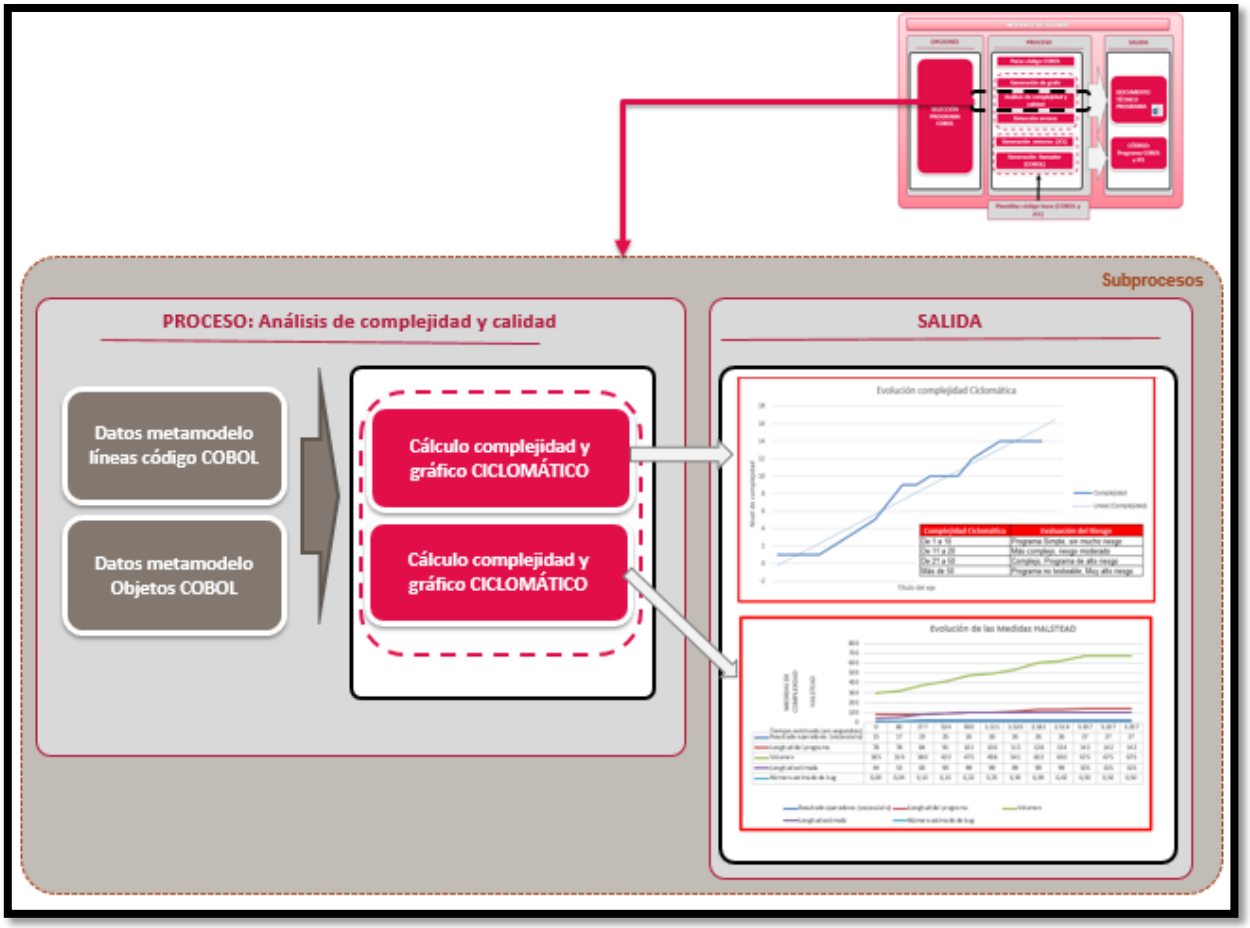



*Figura 31. Arquitectura complejidad y calidad*

Complejidad ciclomática.

Definición en apartado 2.4 del documento. Se definen 4 tipos de complejidad, definidas en la tabla 63:

| Complejidad Ciclomática | Evaluación del Riesgo |
|---|---|
| De 1 a 10 | Programa Simple, sin mucho riesgo |
| De 11 a 20 | Más complejo, riesgo moderado |
| De 21 a 50 | Complejo, Programa de alto riesgo |
| Más de 50 | Programa no testeable, Muy alto riesgo |

*Tabla 63. Niveles complejidad ciclomática*

Se define un gráfico de evolución de la complejidad relacionado con la evolución del código del programa, utilizando para ello dos ejes:

- Eje horizontal, para medir la complejidad ciclomática, empezando de 0 en el nivel inferior.
- Eje vertical, evolución del código.

Ejemplo deevolución ciclomática en figura 32.

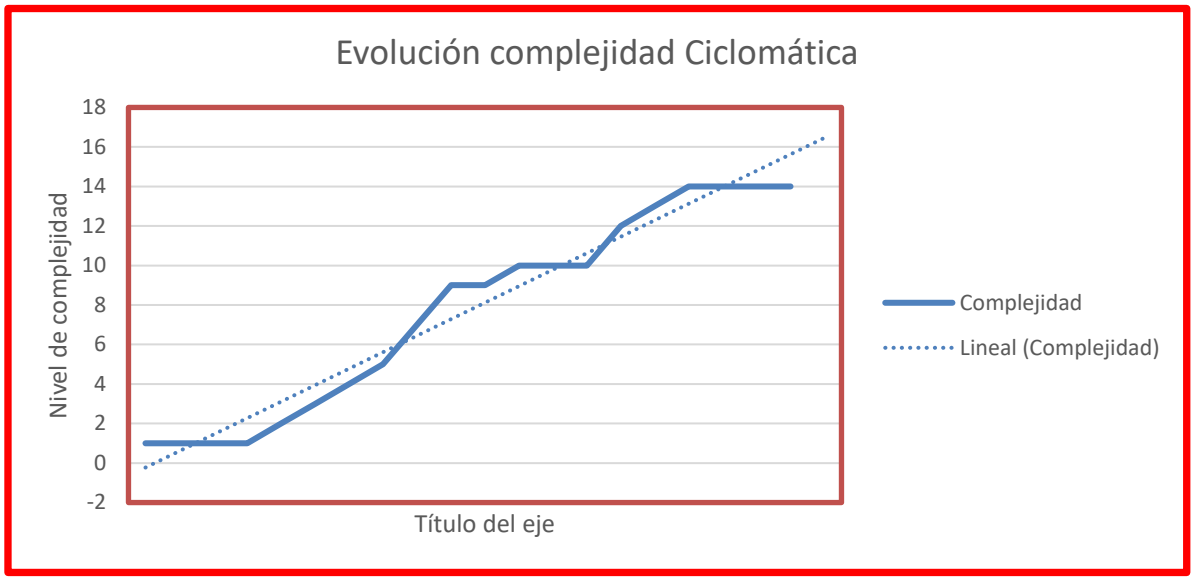



*Figura 32. Gráfico evolución Ciclomática*

Medidas HALSTEAD.

En el apartado 2.4 del documento se describe la definición de estas medidas de calidad. En la tabla 64 se describen todas las variables que intervienen en el cálculo.

| Descripción | Fórmula de cálculo |
|---|---|
| Número de operadores distintos | Este dato se obtiene con un algoritmo de búsqueda dentro de la estructura de líneas de código COBOL. |
| Numero de variables/operandos distintos | Dato obtenido por muestreo en la estructura de objetos COBOL. |
| Resultado operadores (vocabulario) | n=n1+n2 |

| | |
|---|---|
| Total de operadores | Dato obtenido por muestreo en estructura de líneas de código COBOL |
| Total de variables/operandos | Dato obtenido por muestreo en estructura de líneas de código COBOL |
| Longitud del programa | N=N1+N2 |
| Longitud estimada | LE = n1*log2n1+n2*log2n2 |
| Volumen | V=N*log2n |
| Dificultad | D=n1/2*N2/n2 |
| Esfuerzo estimado | E=D*V |
| Tiempo requerido (seg) | T=E/18 |
| Número estimado de bug | B=(E^2/3) / 3000 |
| Número estimado de bug | B=V / 3000 |

*Tabla 64. Campos medidas de calidad*

**Funcionalidad: Detección de errores.**

Se incluirán unas series de validaciones y verificaciones para determinar errores severos y errores leves, los cuales serán informados al usuario dentro del documento de análisis generado.

En la tabla 65 se enumeran los errores más comunes en la programación COBOL los cuales serán analizados y clasificados en la salida del documento:

| Objeto afectado | Descripción del error | Impacto | Clasificación |
|---|---|---|---|
| VARIABLES | Variables definidas y no utilizadas | Posibles problemas por funcionalidades incompletas en su no uso o código basura | WARNING |
| VARIABLES | Uso de cantidades mayores a | Mal funcionamiento en proceso con | SEVERO |

| | | | |
|---|---|---|---|
| | la capacidad de la variable | resultado imprevisibles | |
| ALTERNATIVA | No cerrar condición | Resultado imprevisibles en condiciones. | WARNING |
| BUCLES | No actualizar las variables incluidas en la condición | Creación de bucles sin fin | SEVERO |
| VARIABLES | Movimiento entre variables con diferentes tipos | Cancelación del programa por problemas de datos (OC7) | SEVERO |
| FICHEROS | No usar el fichero | Olvido de funcionalidades | WARNING |
| ALTERNATIVA | Uso de condiciones con literales o números con más capacidad a los campos comparados | Resultado imprevisibles no deseados | SEVERO |
| ALTERNATIVA | Comparación de tipos de datos distintos | Problemas de cancelación o funcionamiento imprevisible | SEVERO |
| FICHEROS | Fichero no cerrado | Los datos del fichero se pueden degradar al finalizar la ejecución | SEVERO |

*Tabla 65. Errores comunes software Mainframes*

Estos errores pueden corresponden a un 70%-80% de los bug de problemas que ocurre en software desarrollado en COBOL para Mainframes.

**Funcionalidad: Generación automática de código y entorno de pruebas (JCL y COBOL).**

Para esta funcionalidad se utilizará un método de **generación de código mixto y generación de código directo.**

Para la generación de código mixto se utilizará una plantilla con el código fuente base y determinadas partes se irán sustituyendo por el código variable.

En la generación directa del código, esta generación se realizará en su totalidad en tiempo de ejecución del programa sin la necesidad de inyección de código a través de plantillas, tal como se observa en la figura 33.

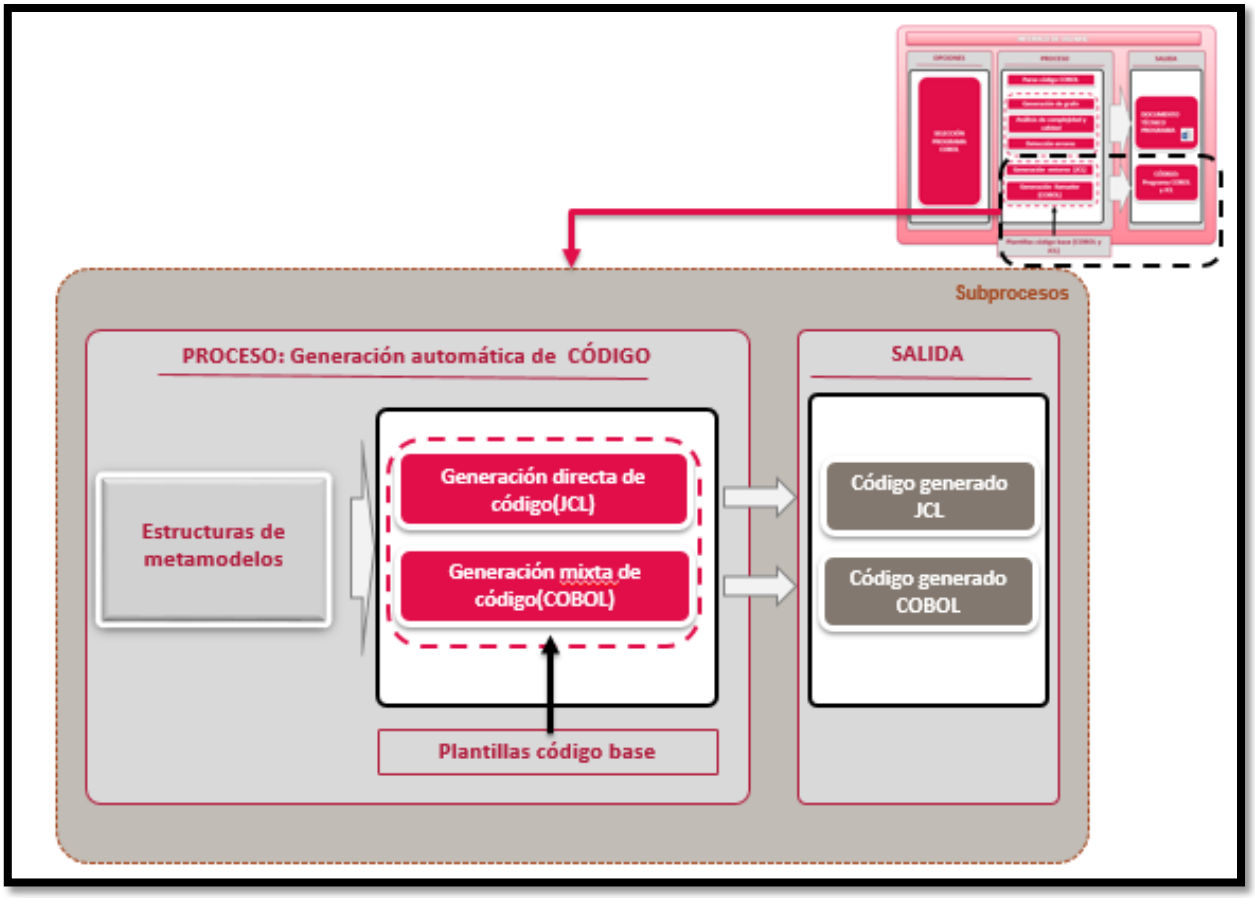



*Figura 33. Arquitectura GAC (COBOL y JCL)*

Generación entorno JCL.

Se utilizará la metodología de generación de código directo para para la obtención del entorno de pruebas JCL. La estructura del JCL se refleja en la tabla 65.

| Estructuras//fichas | Descripción |
|---|---|
| **Cabecera** | Primera línea del código JCL donde se informan de los siguientes campos: Nombre del JOB, clase de ejecución, aplicación, usuario, etc. |
| **Librerías** | Librerías donde están ubicados los objetos a ejecutar. |
| **Pool** | Definición de las salidas de ejecución y visualización del resultado del JOB |
| **Ficheros E** | Definición de los ficheros de entrada. |
| **Ficheros S** | Definición de los ficheros de salida; DSN, longitudes, |

| | |
|---|---|
| | formatos, etc. |
| **Ficha ejecución** | Información sobre el programa que va a ser ejecutado y librerías en caso de cancelamiento |

*Tabla 66. Estructura código JCL*

Ejemplo de JCL:

```
//JCLFXP19 JOB IT,'PRUEBAS UNITARIAS',MSGCLASS=X,CLASS=D,NOTIFY=&SYSUID (línea cabecera)
//*********************************************************************
//*        PRUEBAS UNITARIAS DEL PROGRAMA: FXP19318                   *
//*********************************************************************
//PASO0001 EXEC  PGM=IKJEFT1B,COND=(4,LT)                    (línea cabecera)
//STEPLIB  DD  DSN=BP.OBJETOS.DESARROLLO,DISP=SHR            (librerías)
//         DD  DSN=BP.OBJETOS.PRODUCCION,DISP=SHR            (librerías)
//SYSPRINT DD  SYSOUT=*                                      (pool)
//SYSTSPRT DD  SYSOUT=*                                      (pool)
//SYSOUT   DD  SYSOUT=*                                      (pool)
//SYSDBOUT DD  SYSOUT=9,HOLD=YES,FCB=S800                    (pool)
//SYSABOUT DD  SYSOUT=9,HOLD=YES,FCB=S800                    (pool)
//SYSUDUMP DD  SYSOUT=9,HOLD=YES,FCB=S800                    (pool)
//FENTRADA DD  DSN=TST2.KT.ES.FICHERO.FENTRADA,              (Ficheros E)
//             DISP=SHR
//FENTRAD2 DD  DSN=TST2.KT.ES.FICHERO.FENTRAD2,              (Ficheros E)
//             DISP=SHR
//FSALIDA DD  DSN=TST2.KT.ES.FICHERO.FSALIDA,                (Ficheros S)
//             DISP=(,CATLG,DELETE),                         (Ficheros S)
//             SPACE=(500,(10,1),RLSE),AVGREC=K,             (Ficheros S)
//             LRECL=500,RECFM=FB                            (Ficheros S)
//FSALIDA2 DD  DSN=TST2.KT.ES.FICHERO.FSALIDA2,              (Ficheros S)
//             DISP=(,CATLG,DELETE),                         (Ficheros S)
//             SPACE=(0,(10,1),RLSE),AVGREC=K,               (Ficheros S)
//             LRECL=0,RECFM=FB                              (Ficheros S)
//SYSTSIN  DD  *                                  (Ficha ejecución)
 DSN SYSTEM(DBD1)                                 (Ficha ejecución)
  RUN PROGRAM(FXP19318) PLAN(DESA)                (Ficha ejecución)
      WHEN SYSRC(NE 0) CALL 'BP.OBJETOS.PRODUCCION(CANCELAR)'  (Ficha ejecución)
      END
```

- Líneas '//*' corresponden a literales no código
- (xxxxxxxxxxxxx), Información de la estructura relacionada.
- xxxxx. Información variable según datos de transformación.
- Líneas repetitivas según número de objetos

Para obtener el código de salida JCL se partirá de la estructura de Objetos COBOL, para poder componer las líneas variables del JCL. Los tipos de objetos necesarios para la composición del código son los siguientes:

- **Nombre del programa.** Objeto que permitirá componer el nombre del JOB y programa lanzador en la ficha de ejecución.
- **Ficheros entradas**. Composición de las líneas Ficheros E.
- **Ficheros salida.** Composición de las líneas Ficheros S.

Generación llamador (COBOL).

Utilización de código mixto, para la cual se creará una plantilla con el código base del programa, con código fijo y código variable.

El código generado estará formateado con una **metodología estructurada**, definiendo los siguientes párrafos variables que surgen de las necesidades funcionales. En la tabla 67 se definen esta estructura principal del COBOL.

| Párrafo | Descripción |
|---|---|
| **Identificación** | Identificación del programa principal y generación de variables necesarias. |
| **Conexión** | Generación del área de datos de intercambio. |
| **Trabajo** | Generación de código para la utilización de variables de trabajo. |
| **Llamada** | Generación de la llamada principal al módulo con la estructura de datos correspondiente |
| **Visualización** | Log sobre los datos utilizados en el intercambio |
| **Validación** | Párrafos para validar los campos de entrada y los correspondientes según la opción elegida. |

*Tabla 67. Estructura principal programa COBOL*

Se utilizará la generación mixta, para ello el código base corresponde con el archivo **'COB codigo base.txt' '** en el anexo.

El archivo contiene el siguiente formato por línea:

**NNN**. 3 posiciones de código numérico para indicar la acción que se debe realizar (la 000 indica escritura directa sin generar código, el resto de valores se pueden ver en apartado

de pseudocódigo, pero indicarán la generación de los párrafos de altas, bajas, validaciones, etc).

**Código**. Desde la posición 4 hasta el final de línea el código en COBOL.

**Funcionalidad: Generación automática de documentación.**

El documento técnico, será un archivo tipo Word, siguiendo la estructura definida en la tabla 68.

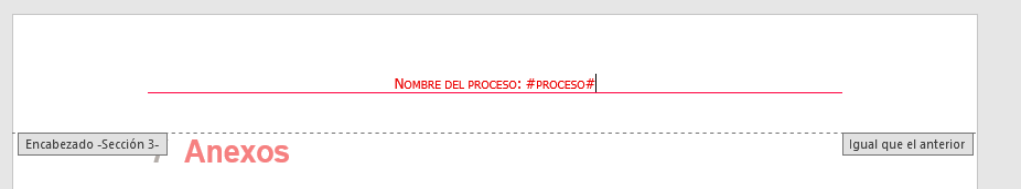
| Índice |
|---|



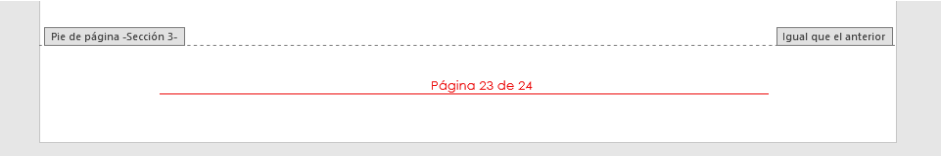
| | FORMATOS |
|---|---|
| **Formato títulos:** Arial 18 color rojo | |

**Formato texto:** Arial 12 color gris claro.

**Cabecera:** Tahoma 9 normal color rojo. Con el siguiente formato:




**Pie**: Century 9 normal color rojo, con formato:




| | |
|---|---|
| | **Hoja de control** |

**El formato será el siguiente:**

## Hoja de control del documento

**Control de la Plantilla**

*Información a editar únicamente cuando se realice un cambio de la plantilla.*

| Versión | Fecha de publicación | Descripción de los cambios |
|---|---|---|
| 1.0 | 19/04/2019 | N/A |

**Control del Entregable**

*Información a editar por el responsable del entregable.*

**Responsable:**

Abel García Sánchez

**Destinatarios:**

- Laboratorio de desarrollo

| Versión | Fecha (*) | Descripción de los cambios |
|---|---|---|
| 1.0 | #fecha# | Versión Inicial |
| | | |

* NOTA 1: *La fecha de la portada habrá de coincidir con la fecha de la última versión incluida en la tabla.*

*NOTA 2: El texto en cursiva del presente documento representa instrucciones para completar cada capítulo por lo que debe sustituirse por el contenido que corresponda en cada caso.*

*La letra con la que se complete cada capítulo ya no será cursiva.*

| Apartado 1 | PROPÓSITO DEL DOCUMENTO |
|---|---|

**Insertar el siguiente literal:**

El presente documento tiene por objeto describir el Planteamiento técnico del diseño de la solución informática correspondiente al objeto COBOL: **#PROGRAMA# ,**para la capa de desarrollo en el entorno Mainframes.

También se incorpora una serie de análisis para medir la calidad y la complejidad de la codificación y una detección temprana de bug potenciales.

| Apartado 2 | ARQUITECTURA DE COMPONENTES (HW Y SW) |
|---|---|

**Insertar el siguiente literal y tabla:**

A continuación, se enumeran los componentes afectados en la solución de enfoque táctico.

Todo el desarrollo se realiza sobre la instalación existente, sin cambios en la arquitectura base.

| ELEMENTO | TIPO | DEFINICIÓN |
|---|---|---|
| #elem1# | #tipo1# | #descrip1# |
| #elem2# | #tipo2# | #descrip2# |
| #elem3# | #tipo3# | #descrip3# |
| … | … | … |
| #elem100# | #tipo100# | #descrip100# |

| Apartado 3 | DIAGRAMA DE EJECUCIÓN |
|---|---|

**Insertar el siguiente literal y tabla:**

En el siguiente diagrama se muestra el flujo de información y de ejecución de la información del programa: **#PROGRAMA#.**

1 <tabla para insertar diagrama de ejecución>

| Apartado 4 | DESCRIPCIÓN DETALLADA DE COMPONENTES |
|---|---|

### 4.1 Ficheros de E/S

**Información detallada sobre los ficheros de ENTRADA al programa:**

#ficherosentrada#

**Información detallada sobre los ficheros de SALIDA al programa:**

#ficherossalida#

### 4.2 Módulos utilizados

**Se utilizan los siguientes módulos:**

#modulos#

| Apartado 5 | GRAFO DE CAMINOS |
|---|---|

<insertar gráfico de caminos>

**A continuación, se muestran las instrucciones de código que generan nuevos caminos:**

#caminos3#

#caminos6#

| Apartado 6 | ANÁLISIS DE CALIDAD Y COMPLEJIDAD DEL PROGRAMA |
|---|---|

**Insertar los siguientes literales y tablas:**

**Complejidad ciclomática**

La Complejidad Ciclomática es una métrica del software en ingeniería del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje

En la siguiente gráfica se muestra la evolución de la complejidad ciclomática dentro de la codificación del programa #PROGRAMA#:

<Incorporar gráfico>

**La complejidad ciclomática del programa es: #ciclomatica#**

**Métricas HALSTEAD**

Métricas de software introducidas por Maurice Howard Halstead en 1977 como parte de su Tratado sobre el establecimiento de una ciencia empírica del desarrollo de software.

Estas métricas se calculan estáticamente a partir del código y su objetivo es el de identificar las propiedades medibles del software y las relaciones entre ellas.

En la siguiente tabla se describen los campos utilizados y su cálculo:

| Campo | Descripción | Fórmula de cálculo |
|---|---|---|
| **n1** | Número de operadores distintos | Dato que se obtiene del programa |
| **n2** | Numero de variables/operandos distintos | Dato que se obtiene del programa |
| **ETC** | ETC | ETC |

Los resultados obtenidos después del análisis son los siguientes:

<Incorporar gráfico>

Las evoluciones de las medidas según desarrollo de la codificación se muestran en la siguiente gráfica:

<Incorporar gráfico>

| Apartado 7 | DETECCIÓN TEMPRANA DE BUG |
|---|---|

En este apartado se realiza un análisis de detección de posibles errores leves o potenciales que pueden derivar a bug en tiempo de ejecución del programa.

**Errores leves detectados:**
#erroresleves1#
#erroresleves2#
#erroresleves3#
...
#erroresleves40#

**Errores graves detectados:**
#erroresgraves1#
#erroresgraves2#
#erroresgraves3#
...
#erroresgraves40#

| Apartado 8 | CASOS DE PRUEBAS UNITARIAS |
|---|---|

Existen muchos tipos de pruebas unitarias, pero si tomamos como referencia el código, podemos hacer dos grandes divisiones; las pruebas de caja blanca [44] u orientadas al código que se desea probar o un segundo enfoque orientado a las funcionalidades del software, sin llegar a conocer el código, llamadas preubas de caja negra. Para la generación automática se utilizarán las primeras, actuando sobre el código.

En estas pruebas unitarias generadas automáticamente, se generarán todos los

posibles caminos, existiendo dos criterios de cobertura [45], uno de ellos correspondería a la ejecución de todas las pruebas y el otro criterio sería a base de combinaciones llamado MCMG "cobertura mínima de gráfico mixto".

Los literales que aparecerán en el documento serán los siguientes:

*En este apartado se describen los casos de pruebas unitarias recomendables para validar el programa analizado.*

*La siguiente tabla contiene un resumen de los casos de pruebas recomendados y su estado:*

**Resumen de casos de prueba:**

| ID PRUEBA | DESCRIPCIÓN | SITUACIÓN |
|---|---|---|
| Caso-V-1 | Prueba del camino: 1 Instrucción: PROCEDURE | Pendiente |
| Caso-V-2 | Prueba del camino: 2 Instrucción: DISPLAY "ES IGUAL A 25" | Pendiente |
| Caso-V-3 | Prueba del camino: 3 Instrucción: DISPLAY "ES DISTINTO A 25" | Pendiente |
| Caso-V-4 | Prueba en vacío: Realizar una prueba sin información en las entradas externas | Pendiente |
| Caso-V-5 | Prueba estrés: Inclusión de entradas externas con mucha información | Pendiente |
| Caso-V-6 | Pruebas búsqueda límites: Pruebas para buscar límites en bucles y condiciones | Pendiente |

| Apartado 9 | DOCUMENTOS RELACIONADOS |
|---|---|

**Insertar los siguientes literales y tablas:**

*Se incluirá en este apartado toda la documentación a la que se haga referencia a lo largo de este documento*

*La utilización de este capítulo es OPCIONAL.*

| Ref. | Código | Nombre |
|---|---|---|
| | | |
| | | |
| | | |
| | | |

| Apartado 10 | ANEXOS |
|---|---|

| |
|---|
| Espacio reservado para la inclusión de anexos. |
| |

*Tabla 68. Formato documento DT COBOL*

**PSEUDOCÓDIGO del módulo 'GenerarAnalisisCOBOL'.**

| Párrafo principal |
|---|
| <Inicializar_Datos> 'Iniciar las variables utilizadas en el proceso<br><Parse_codigo_cobol > 'Obtención metamodelos (análisis semántico)<br><Crear_grafo_caminos> ' Obtener los distintos caminos de ejecución del código<br><Grafico_ciclomatico> ' Generar diagrama ciclomático con los distintos caminos<br><Calcular_metrica_HALSTEAD> 'Obtener datos de calidad<br>< Generar_JCL> ' Generación automática del entorno para pruebas<br><Generar_COBOL_principal> 'Obtención del programa llamdor<br>< Generar_diagrama> 'Generación del diagrama de objetos del programa<br>< Visualizar_Arrays> 'Log sobre la ejecución<br>< Generar_Doc_Analisis> ' Generación del documento de análisis |
| Parse_codigo_cobol |
| <conexion_BD><br><guardar_procesos><br><generar_temporal> 'generamos un código intermedio desplegando procesos<br><analizar_cobol_crear_metamodelo> 'parse para obtener datos para modelos<br><cerrar_conexion_BD> |
| Crear_grafo_caminos |
| <preparar_hoja_grafico><br>Hasta fin_instrucciones<br>Si inicio <dibujar inicio> finsi<br>Si cambio_camino<br><guardar_posiciones><br><dibujar_rectangulo><br><enlazar><br>finSI |

```
Fin hasta
<dibujar terminador>
```

Grafico_ciclomatico

```
<iniciar_variables>
<preparar_gráfica>
unidad = contador de líneas de codigo / 20 ‘obtener unidad para calcular gráfica
                                                ‘ de la evolución ciclomática
For i=1 to 20
   <obtener_camino>
   <informar_camino>
Next i
```

Calcular_metrica_HALSTEAD

```
<iniciar variables>
<obtención_variables_HALSTEAD>
<informar_variables_en_celdas>
<Obtención de gráfica>
```

Generar_JCL

```
<preparar_salida _JCL> ‘crear carpeta de salida y fichero para guardar JCL
<componer_cabecera>  ‘generar la cabecera del JCL
<componer_librerias_ejecución>
<componer_fichas_visualización>
<componer_fichas_ficheros_E>
<componer_fichas_ficheros_S>
<componer_ficha_ejecución>
<cerrar_fichero>
```

Generar_COBOL_principal

```
<preparar_salida_COBOL>
<abrir_fichero_codigo_base>
<leer_codigo_base>
Hasta fin fichero base
    Evaluar código_entrada
    Caso "000" 'Se escribe el código base de la plantilla
        registro_cobol = VBA.Mid(FENTRADA, 4)
            Caso "001" 'sin utilización
        salida_nombre_programa_cobol
    Caso "002" 'Generación del código variables que serán utilizadas
        aux_situacion_variable = "AUX-"
        <generar_variables_cobol>
    Caso "003" 'creación de los campos de entrada al módulo
```

```
        aux_situacion_variable = "LNK-"
        <generar_variables_cobol>
    Caso "004" 'Validación de los campos de entrada
        <validar_entrada_cobol>
    Caso "005" 'llamar al modulo para hacer pruebas
        <Llamar_modulo_cobol>
    Caso "006" 'llamar al modulo para hacer pruebas
        <crear_procedure_cobol>
    Case "007" 'mover a variables trabajo
        <mover_a_variables_trabajo_cobol<
    Case "008" 'mover a variables de trabajo para (no es recomendable utilizar las
de linkage
        <visualizar_variables_entrada_cobol<
    OTROS  <dar-error>
    FIN-EVALUAR
  <escribir_salida_cobol>
   <leer_codigo_base>

Fin Hasta
Cerrar ficheros I/O.
```

Generar_diagrama

```
<preparar_hoja>
<dibujar_programa_principal>  'Rectángulo con el progama principal
Recorrer estructura objetos
  Si objeto_fichero_E <dibujar_ficheros_E> 'Dibujar los ficheros de Entrada
                      <enlazar_objetos>
  Si objeto_fichero_S <dibujar_ficheros_S> 'Dibujar los ficheros de Salida
                      <enlazar_objetos>
  Si objeto_modulo <dibujar_modulo> 'Dibujar llamada modulos
                      <enlazar_objetos>

Fin recorrer
```

Visualizar_Arrays

```
<visualizar_estructura_objetos>
<visualizar_estructura_código>
<visualizar_procesos>
```

Generar_Doc_Analisis

```
<inicializar_objeto_Word>
```

<seleccionar_plantilla >
<sustituir variables > ‘sustitución de todas las variables de finidas en la plantilla
‘por los datos del metamodelo (programas, pasos, ficheros, etc)
<generar_apartado_arquitectura> ‘sustitución de varibles de objetos
<generar_apartado_diagrama> ‘copiar imagen del diagrama generado en apartado
<generar_apartado_detalle_componentes> ‘Objetos e/s, módulos,etc
<generar_apartado_grado> ‘copia de la imagen del grafo de caminos y
detalle de los caminos
<generar_apartado_complejidad_ciclomática> ‘Generación del gráfico de evolución
ciclomática y el grado de complejidad
<generar_apartado_métricas_HALSTEAD> ‘copia del gráfico HALSTEAD y
‘la tabla de valores
<generar_apartado_detección_errores> ‘visualización de los errores leves y
Graves que han sido localizados en el
Análisis
<generar_apartado_pruebas_unitarias> ‘Recomendación de las pruebas unitarias
Que se deben realizar
<limpiar hojas y espacios vacíos> ‘se realiza una limpieza de espacios que han
‘ quedado sin rellenar en el documento
<guardar documento> ‘salvar el documento con el nombre
SALIDA__<nombre programa>_1.0.doc

## *3.6 Definición de casos de prueba*

Para la definición de los casos de prueba [46][47] se han tenido en cuenta los requisitos definidos en el desarrollo, para que exista una trazabilidad entre la definición de los especificado en el problema y los resultados finales y se pueda validar de forma clara el requisito inicial.

Este planteamiento de pruebas está basado en la metodología que se ha seguido en el proyecto, es decir la waterfall. Si se hubiera seguido una metodología agile cómo scrum, se deben seguir otros plateamientos [48].

Recordamos los requisitos iniciales en la tabla 69.

| Identificación Requisito | Descripción del requisito | Tipo de requisito |
|---|---|---|
| REQ-001 | Generación automática de documentación técnica sobre procesos JCL. | Funcional |
| REQ-001-01 | Analizador semántico de JCL para obtener la arquitectura de componentes | Funcional |

| | | | |
|---|---|---|---|
| | | software del JCL | |
| | REQ-001-02 | Creación de un diagrama de flujo de datos de ejecución (nivel1) | Funcional |
| | REQ-001-03 | Creación de un diagrama de flujo de datos detallado del JCL (nivel2) | Funcional |
| | REQ-001-04 | Generación de un DT del proceso con un formato predefinido. | Funcional |
| REQ-002 | | Generación automática de entornos de prueba Mainframes y analizador de código COBOL | Funcional |
| | REQ-002-01 | Generación del código COBOL del programa principal para pruebas. | Funcional |
| | REQ-002-02 | Generación del entorno de pruebas del lanzador para entornos MAINFRAMES (generación del JCL).. | Funcional |
| | REQ-002-03 | Medición de la calidad y complejidad del código a través de métodos empíricos | Funcional |
| | REQ-002-04 | Detección automática de fallas o posibles problemas potenciales en el programa a través de muestreos | Funcional |
| | REQ-002-05 | Crear un grafo con el flujo de caminos que puede tener el programa. | Funcional |
| | REQ-002-06 | Generación automática del DT del programa con el formato predeterminado. | Funcional |
| REQ-003 | | Facilidad de uso | No funcional |
| REQ-004 | | Manual de usuario | No funcional |
| REQ-005 | | Rendimiento óptimo | No funcional |
| REQ-006 | | Seguimiento del estado del proceso | No funcional |

*Tabla 69. Requisitos para pruebas*

Los casos de prueba serán agrupados en los siguientes bloques

- **Validación de sub requisitos funcionales.** Validación de los sub requisitos definidos con la definición de pruebas unitarias. Definidos con CASO-U-nnn
- **Validación de Requisitos funcionales principales.** Validación de los requisitos principales con pruebas íntegras. Definidos con CASO-I-nnn
- **Validación de Requisitos no funcionales.** Validación de los Otros requisitos no funcionales del proyecto. Definidos con CASO-O-nnn
- **Fallas**. Casos de prueba con identificación CASO-F-nnn.

En la tabla 70 se definen los casos de prueba

| Id. Prueba | Descripción de la prueba |
|---|---|
| CASO-U-001 | **Validación sub requisito REQ-001-01.** Verificación de la correcta extracción de los elementos software de un proceso JCL. |
| CASO-U-002 | **Validación sub requisito REQ-001-02.** Comprobar el correcto diseño del diagrama de ejecución, el DFD nivel 1. |
| CASO-U-003 | **Validación sub requisito REQ-001-03.** Comprobar el correcto diseño del diagrama detallado, el DFD nivel 2. |
| CASO-U-004 | **Validación sub requisito REQ-001-04.** Creación del documento DT según plantilla y nombre especificado. |
| CASO-U-005 | **Validación sub requisito REQ-001-01.** Verificación de la correcta extracción de los elementos software de un proceso JCL. |
| CASO-I-001 | **Validación del requisito REQ-001.** Generación automática de un documento DT con los apartados y características definidas en el planteamiento de la solución del requisito REQ-001 |
| CASO-I-002 | **Validación del requisito REQ-001.** Generación automática de un documento DT con los apartados y características definidas en el planteamiento de la solución del requisito REQ-001 con un JCL más grande |
| CASO-O-001 | **Validación sub requisito REQ-003.** Comprobar la facilidad de uso de la herramienta para el REQ-001 |
| CASO-O-002 | **Validación sub requisito REQ-005.** Verificar el tiempo del proceso. |
| CASO-O-003 | **Validación sub requisito REQ-006.** Verificar el avance el proceso y su correcto entendimiento. |
| CASO-O-004 | Verificación del correcto funcionamiento de PARSE en código COBOL |
| CASO-U-006 | **Validación sub requisito REQ-002-01.** Generar programa llamador en COBOL. |
| CASO-U-007 | **Validación sub requisito REQ-002-02.** Generar entorno de pruebas para mainframes (obtener JCL). |
| CASO-U-008 | **Validación sub requisito REQ-002-03.** Medición de la calidad y complejidad del código a través de métodos empíricos |
| CASO-U-009 | **Validación sub requisito REQ-002-03.** Medición de la calidad y complejidad del código a través de métodos empíricos. (con información más compleja) |
| CASO-U-010 | **Validación sub requisito REQ-002-04.** Detección automática de fallas o posibles problemas potenciales en el |

| | |
|---|---|
| | programa a través de muestreos |
| CASO-U-011 | **Validación sub requisito REQ-002-05.** Crear un grafo con el flujo de caminos que puede tener el programa. |
| CASO-U-012 | **Validación sub requisito REQ-002-06.** Generación automática del documento según formato esperado. |
| CASO-I-003 | **Validación del requisito REQ-002.** Generación automática de un documento de análisis de código COBOL con todos sus apartados-. |
| CASO-I-004 | **Validación del requisito REQ-002.** Generación automática de un documento DT con los apartados y características definidas en el planteamiento de la solución del requisito REQ-001 con un JCL más grande |

*Tabla 70. Definición de casos de prueba*

**Ejecución de las pruebas.**

Ver anexo 'Evidencias ejecución de pruebas'.

# CAPÍTULO 4

# 4. Validación experimental

Con el fin de validar experimentalmente la herramienta desarrollada, se han utilizado dos casos de prueba que se detallan a continuación, con dos ejecuciones de la herramienta para comprobar su funcionamiento.

## *4.1 Caso experimental 1*

Se realizará la validación experimental del requisito REQ-001 (Generación automática de documentación técnica sobre procesos JC) y sus subrequisitos asociados.

En esta ejecución, se seleccionará un archivo que contenga código escrito en JCL y lo convertirá en conocimiento accesible a personas que no cuentan con experiencia en este tipo de lenguajes.

El código JCL utilizado es un job que consta de una cabecera, en la cual se identifica el trabajo y 5 pasos de ejecución, con distintas entradas y salidas. El resultado final debe corresponder con un documento técnico en el cual se debe reflejar toda la arquitectura de elementos de JCL, un diagrama de ejecución de alto nivel, un diagrama detallado con las E/S y el detalle de cada uno de los pasos (en total 5).

En la tabla 71 se refleja el resultado esperado de esta ejecución experimental:

| Elementos de entrada | Generación en salida |
|---|---|
| Arquitectura de componentes | **Se deben obtener los siguientes componentes:**<br>5 Pasos.<br>3 programas y 2 sort.<br>17 ficheros de E/S |
| 5 pasos de ejecución | **Diagrama de ejecución:** |

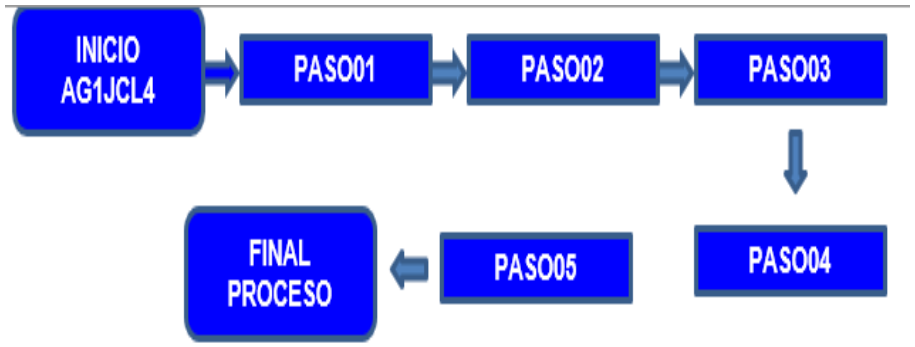
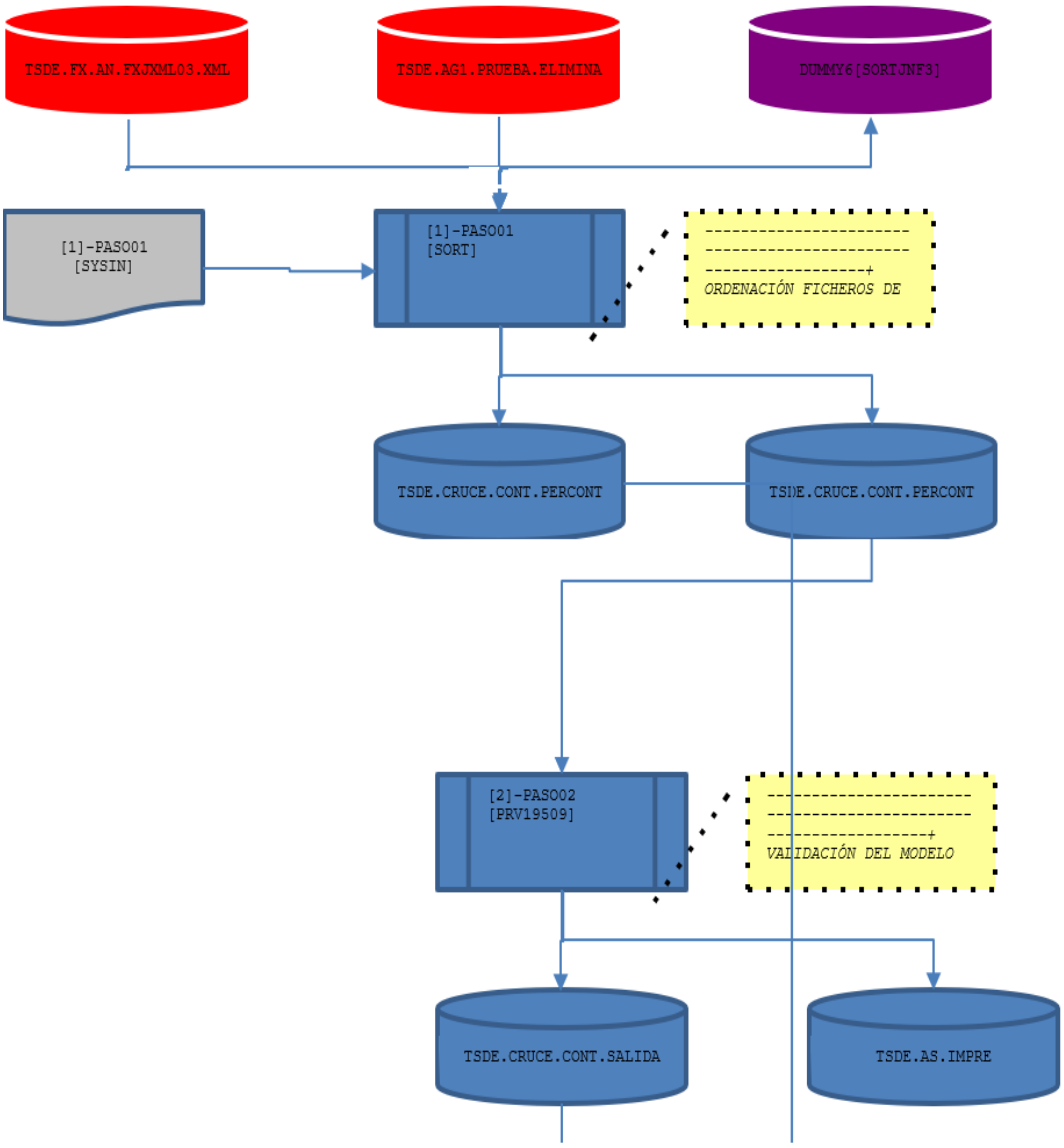
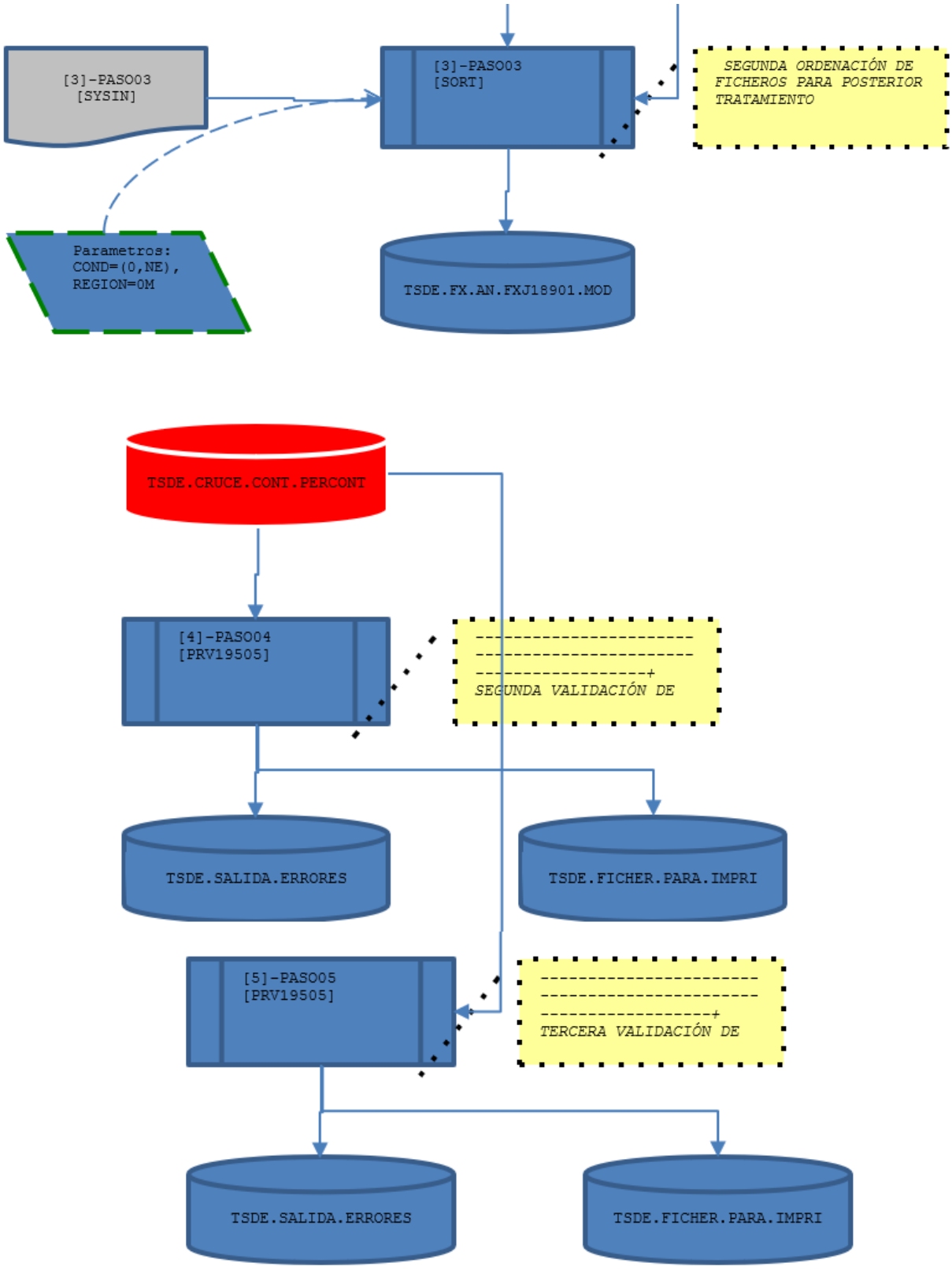



**Diagrama detallado DFD nivel 2:** 5 Pasos, 3 programas/2 sort y 17 ficheros de E/S






| | |
|---|---|
| Comentarios, pasos, etc. | **Detalle de cada uno de los pasos:** |

| | |
|---|---|
| | **Nombre del paso: PASO01.**<br><br>**Identificación:**<br>SORT<br>**Tipo:**<br>Programa o Sort<br>**Nuevo/Modificado/Sin cambios:**<br>Sin cambios<br>**Ficheros de entrada:**<br>SORTJNF1 DSN: TSDE.FX.AN.FXJXML03.XMLM290.P1.DIA30<br>SORTJNF2 DSN: TSDE.AG1.PRUEBA.ELIMINAR<br>SORTJNF3 DSN: DUMMY6[SORTJNF3]<br>SYSIN DSN: [1]-PASO01 [SYSIN]<br><br>**Ficheros de salida:**<br>F1ONLY DSN: TSDE.CRUCE.CONT.PERCONT.F1<br>F2ONLY DSN: TSDE.CRUCE.CONT.PERCONT.F2<br><br>**Comentarios:**<br>----------+ ORDENACIÓN<br>FICHEROS DE ENTRADA AL PROCESO +----------<br>----------+ |

*Tabla 71. Salida esperada en el Caso Experimental 1*

En la tabla 72 se muestran las evidencias de la ejecución experimental 1 con toda la información de entrada y los resultados de salida.

| **ID:** | **CASO-EXPERIMENTAL-001** | **Responsable:** | Abel |
|---|---|---|---|
| **Descripción** | Prueba íntegra de la funcionalidad principal de generación de documentación automática para código JCL. | | |
| **Datos de entrada** | El JCL que se analizará será el siguiente: | | |

```
//AG1JCL4  JOB IT,DESARROLLO,MSGCLASS=X,CLASS=D,NOTIFY=TSDEAG1
//*****************************************************************
//*-----------------------------------------------------------------+
//*  ORDENACIÓN FICHEROS DE ENTRADA AL PROCESO                      +
//*-----------------------------------------------------------------+
//PASO01   EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SORTJNF1 DD DSN=TSDE.FX.AN.FXJXML03.XMLM290.P1.DIA30,DISP=SHR
//SORTJNF2 DD DSN=TSDE.AG1.PRUEBA.ELIMINAR,DISP=SHR
//SORTJNF3 DD DUMMY
//F1ONLY   DD DSN=TSDE.CRUCE.CONT.PERCONT.F1,
//         DISP=(,CATLG,DELETE),
//         SPACE=(20,(9,9),RLSE),STORCLAS=SCDEPER,
//         LRECL=223,RECFM=FB
//F2ONLY   DD DSN=TSDE.CRUCE.CONT.PERCONT.F2,
//         DISP=(,CATLG,DELETE),
//         SPACE=(20,(9,9),RLSE),STORCLAS=SCDEPER,
//         LRECL=072,RECFM=FB
//SYSIN    DD *
//*-----------------------------------------------------------------+
//*  VALIDACIÓN DEL MODELO 195 DE LA AEAT                           +
//*-----------------------------------------------------------------+
//PASO02    EXEC PGM=PRV19509
//STEPLIB   DD  DISP=SHR,DSN=librería de LOAD
//ENTRADA   DD  DSN=TSDE.CRUCE.CONT.PERCONT.F2,DISP=SHR
//SALIDA    DD  DSN=TSDE.CRUCE.CONT.SALIDA,
//          UNIT=3390,DISP=(,CATLG,UNCATLG),
//          DCB=(RECFM=FB,LRECL=418),
```

```
//           SPACE=(CYL,(1,1),RLSE)
//IMPRE      DD  DSN=TSDE.AS.IMPRE,
//           UNIT=3390,DISP=(,CATLG,UNCATLG),
//           DCB=(RECFM=FB,LRECL=133),
//           SPACE=(CYL,(1,1),RLSE)
//SYSOUT     DD  SYSOUT=*
//SYSPRINT   DD  SYSOUT=*
//*********************************************************************
//* SEGUNDA ORDENACIÓN DE FICHEROS PARA POSTERIOR TRATAMIENTO
//*********************************************************************
//*********************************************************************
//PASO03    EXEC  PGM=SORT,COND=(0,NE),
//             REGION=0M
//SYSOUT    DD  SYSOUT=*
//SORTWK01  DD  SPACE=(CYL,(150,10))
//SORTWK02  DD  SPACE=(CYL,(150,10))
//SORTWK03  DD  SPACE=(CYL,(150,10))
//SORTWK04  DD  SPACE=(CYL,(150,10))
//SORTWK05  DD  SPACE=(CYL,(150,10))
//SORTWK06  DD  SPACE=(CYL,(150,10))
//SORTIN    DD  DSN=TSDE.CRUCE.CONT.PERCONT.F1,
//              DISP=SHR
//SORTIN2   DD  DSN=TSDE.CRUCE.CONT.SALIDA,
//              DISP=SHR
//SORTOUT   DD  DSN=TSDE.FX.AN.FXJ18901.MODEL189.ALLIANZ.PAPRU,
//              DISP=(,CATLG,CATLG),STORCLAS=SCDEPER,
//              SPACE=(500,(1,1),RLSE),AVGREC=K,
//              LRECL=500,RECFM=FB
//SYSIN     DD  *
     SORT FIELDS=COPY
//*-------------------------------------------------------------------+
//*  SEGUNDA VALIDACIÓN DE FICHEROS                                   +
//*-------------------------------------------------------------------+
//PASO04    EXEC PGM=PRV19505
//STEPLIB   DD  DISP=SHR,DSN=librería de LOAD
//CINTA     DD  DSN=TSDE.CRUCE.CONT.PERCONT.NEW,DISP=SHR
//SALIDA    DD  DSN=TSDE.SALIDA.ERRORES,
//           UNIT=3390,DISP=(,CATLG,UNCATLG),
//           DCB=(RECFM=FB,LRECL=418),
//           SPACE=(CYL,(1,1),RLSE)
//IMPRE     DD  DSN=TSDE.FICHER.PARA.IMPRI,
//           UNIT=3390,DISP=(,CATLG,UNCATLG),
//           DCB=(RECFM=FB,LRECL=133),
//           SPACE=(CYL,(1,1),RLSE)
//SYSOUT    DD  SYSOUT=*
//SYSPRINT  DD  SYSOUT=*
//*-------------------------------------------------------------------+
//*  TERCERA VALIDACIÓN DE FICHEROS                                   +
//*-------------------------------------------------------------------+
//PASO05    EXEC PGM=PRV19505
//STEPLIB   DD  DISP=SHR,DSN=librería de LOAD
//CINTA     DD  DSN=TSDE.CRUCE.CONT.PERCONT.NEW,DISP=SHR
//SALIDA    DD  DSN=TSDE.SALIDA.ERRORES,
//           UNIT=3390,DISP=(,CATLG,UNCATLG),
//           DCB=(RECFM=FB,LRECL=418),
//           SPACE=(CYL,(1,1),RLSE)
//IMPRE     DD  DSN=TSDE.FICHER.PARA.IMPRI,
//           UNIT=3390,DISP=(,CATLG,UNCATLG),
//           DCB=(RECFM=FB,LRECL=133),
//           SPACE=(CYL,(1,1),RLSE)
//SYSOUT    DD  SYSOUT=*
//SYSPRINT  DD  SYSOUT=*
```

| | |
|---|---|
| **Datos de salida** | Generación de la siguiente información: |

| | |
|---|---|
| **esperados** | • Detalle de elementos del proceso JCL.<br>• Diseño DFD nivel 1 con la ejecución de los distintos pasos que tiene el proceso. En este caso debería dibujar el inicio + 5 procesos + final de proceso.<br>• Diseño DFD nivel 2. Detalle con todos los programas de ejecución, con sus ficheros de entrada y salida. El gráfico debe estar acorde con el código.<br>• Generación del documento DT con las siguientes características:<br>  ○ Formato del documento según especificaciones del diseño.<br>  ○ Relación de la arquitectura de componentes.<br>  ○ Inclusión del gráfico del DFD nivel 1.<br>  ○ Inclusión del gráfico del DFD nivel 2.<br>  ○ Detalle de los pasos del JCL |

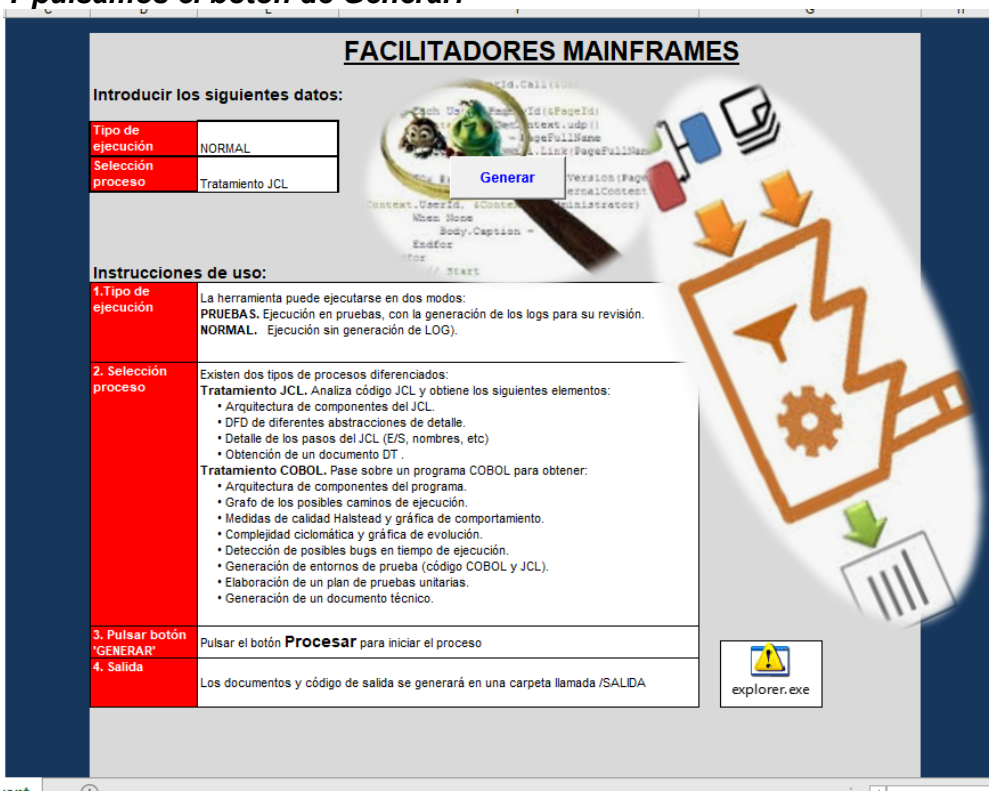
## ANÁLISIS DE LOS RESULTADOS

Entramos a la herramienta:

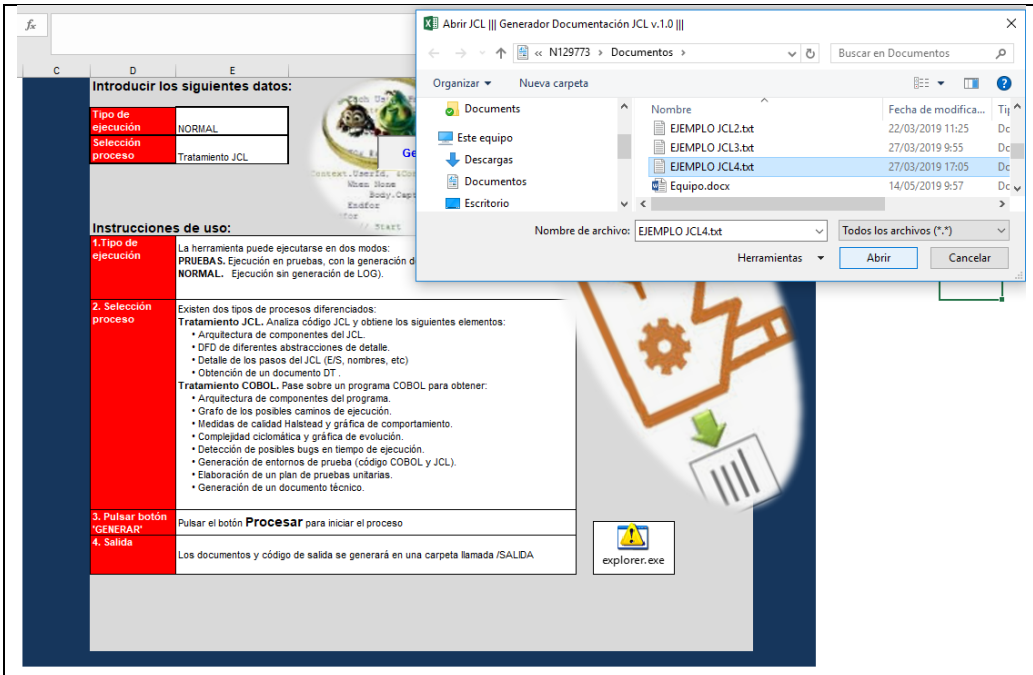
Ejecución de la herramienta y marcamos las siguientes opciones:

| | |
|---|---|
| **Tipo de ejecución** | NORMAL |
| **Selección proceso** | Tratamiento JCL |

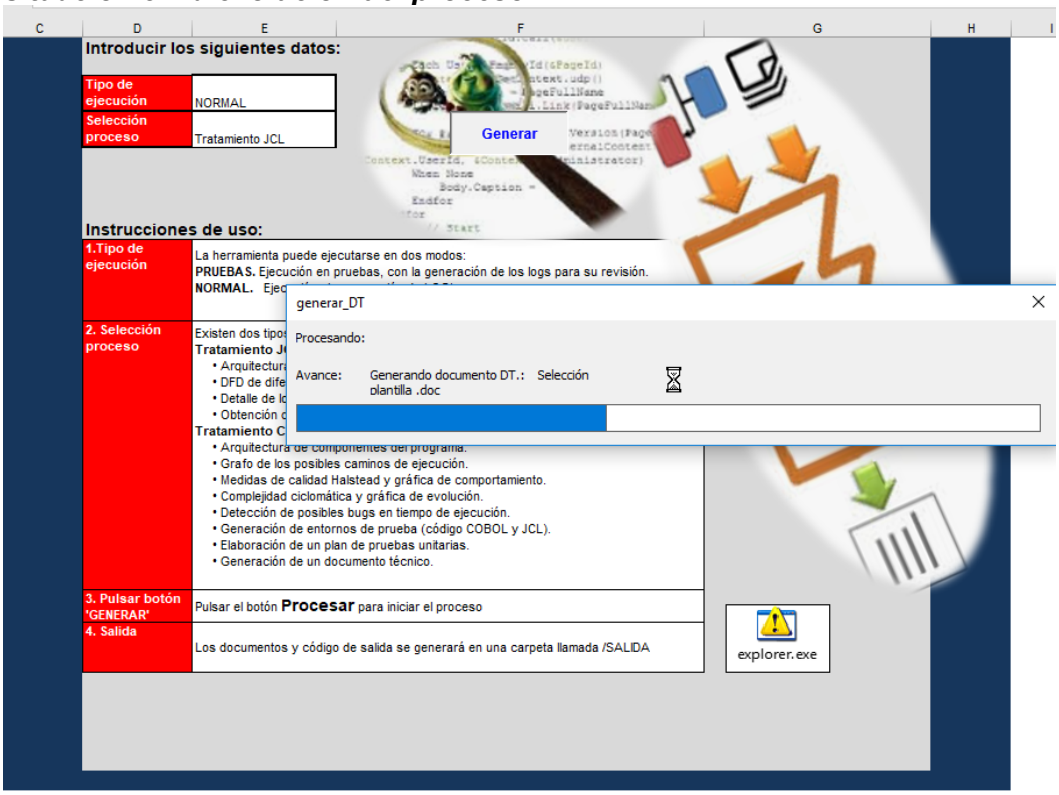
***Y pulsamos el botón de Generar:***



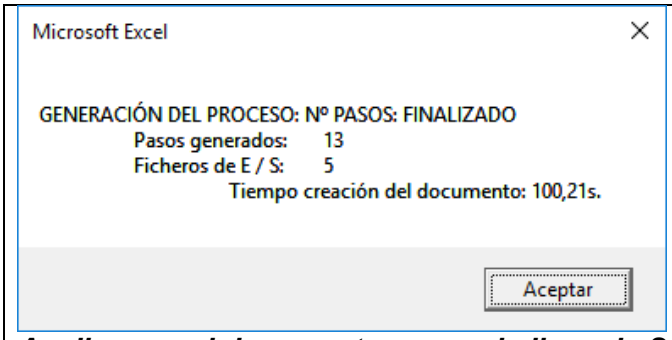

***Se nos abre una ventana emergente para seleccionar el archivo correspondiente al JCL. Lo seleccionamos y le damos a aceptar.***



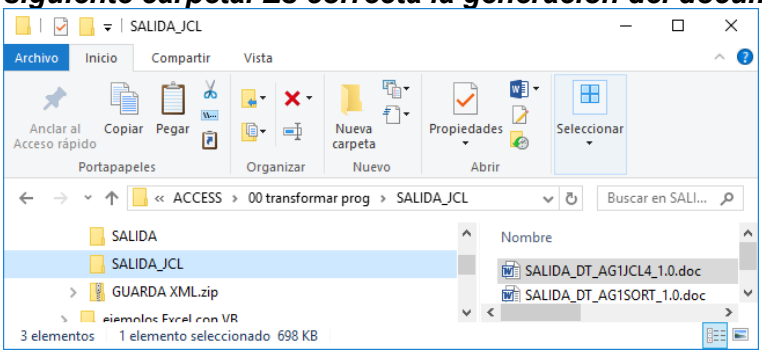

***Se inicia el proceso y nos indica la evolución con una barra de estado. Esta barra indica la situación en la evolución del proceso:***



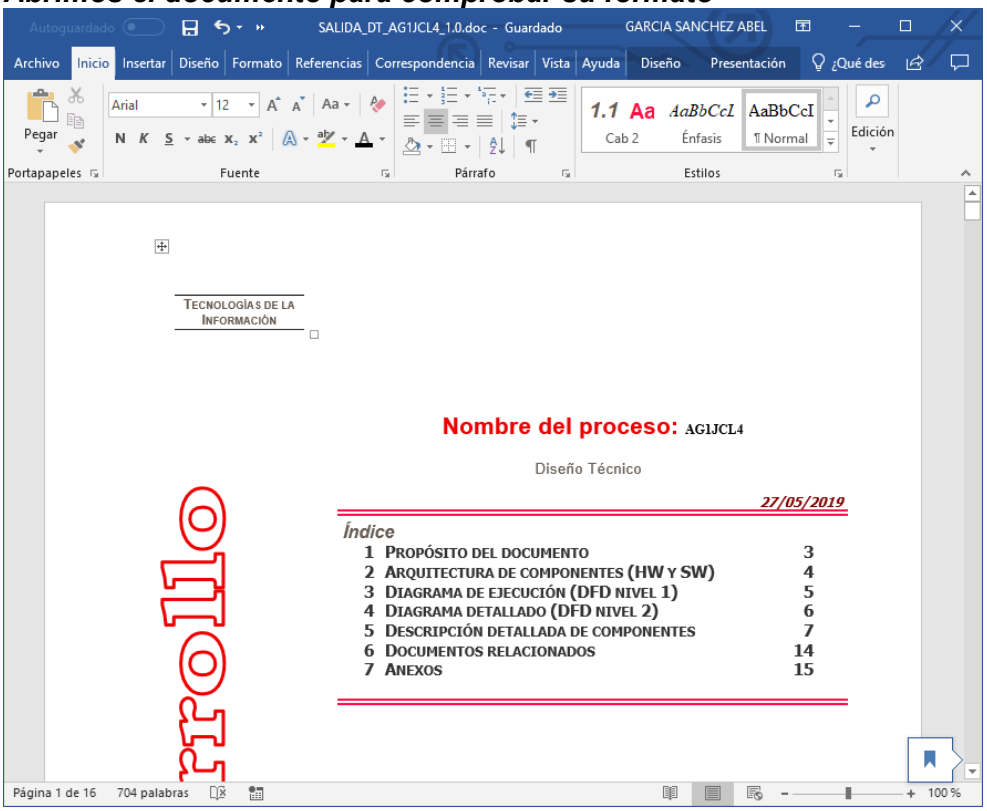

***Una vez finalizado nos muestra unas estadísticas con los pasos generados, ficheros y el tiempo transcurrido en al generación:***




***Analizamos el documento generado llamado SALIDA_DT_AG1JCL4_1.0.doc ubicado en la siguiente carpeta. Es correcta la generación del documento:***




***Abrimos el documento para comprobar su formato***

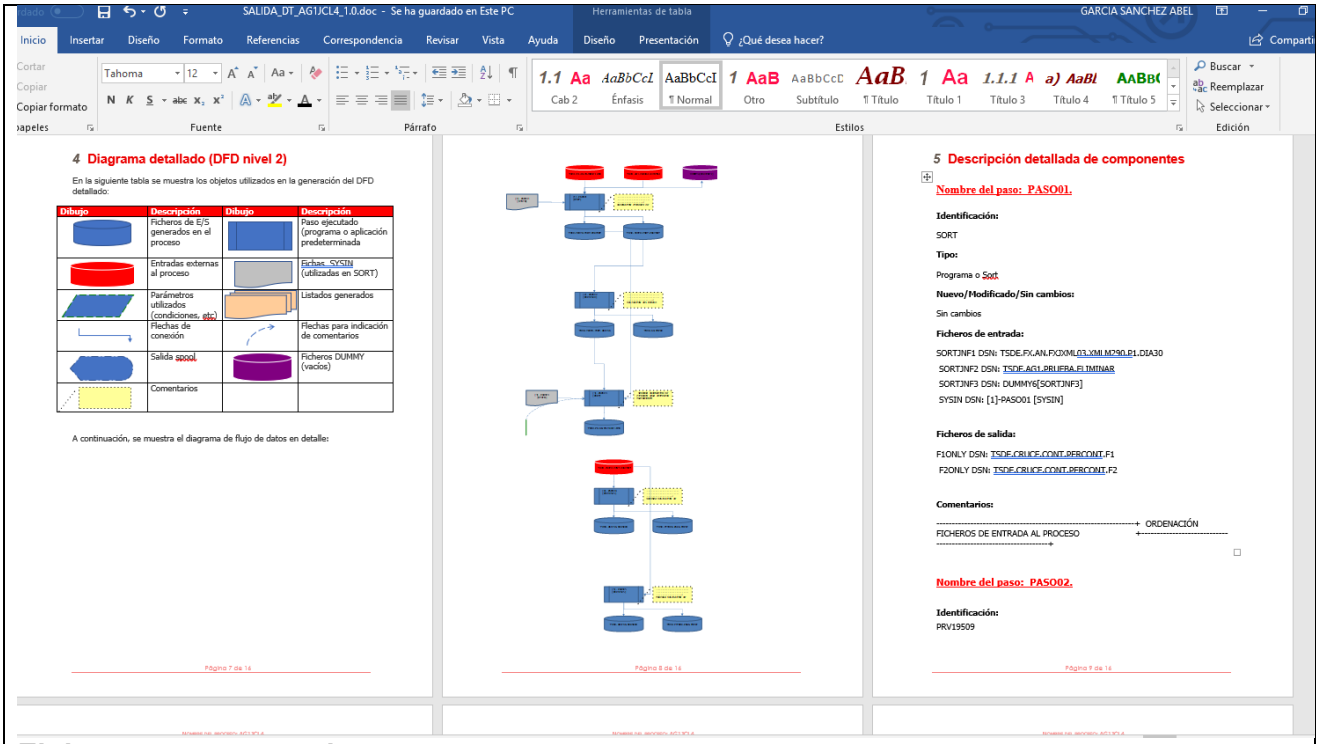



***Revisamos los componentes extraídos y son correctos:***

| ELEMENTO | TIPO | PASO UBICADO | DESCRIPCIÓN |
|---|---|---|---|
| SORT | Programa | PASO01 | Programa |

| | | | |
|---|---|---|---|
| SORTJNF1 | Fichero | PASO01 | Fichero de entrada: TSDE.FX.AN.FXJXML03.XMLM290.P1.DIA30 |
| SORTJNF2 | Fichero | PASO01 | Fichero de entrada: TSDE.AG1.PRUEBA.ELIMINAR |
| SORTJNF3 | Fichero | PASO01 | Fichero de entrada: DUMMY6[SORTJNF3] |
| F1ONLY | Fichero | PASO01 | Fichero de salidaTSDE.CRUCE.CONT.PERCONT.F1 |
| F2ONLY | Fichero | PASO01 | Fichero de salidaTSDE.CRUCE.CONT.PERCONT.F2 |
| SYSIN | Fichero | PASO01 | Fichero de entrada: [1]-PASO01 [SYSIN] |
| PRV19509 | Programa | PASO02 | Programa |
| ENTRADA | Fichero | PASO02 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.F2 |
| SALIDA | Fichero | PASO02 | Fichero de salidaTSDE.CRUCE.CONT.SALIDA |
| IMPRE | Fichero | PASO02 | Fichero de salidaTSDE.AS.IMPRE |
| SORT | Programa | PASO03 | Programa |
| SORTIN | Fichero | PASO03 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.F1 |
| SORTIN2 | Fichero | PASO03 | Fichero de entrada: TSDE.CRUCE.CONT.SALIDA |
| SORTOUT | Fichero | PASO03 | Fichero de salidaTSDE.FX.AN.FXJ18901.MODEL189.ALLIANZ.PAPRU |
| SYSIN | Fichero | PASO03 | Fichero de entrada: [3]-PASO03 [SYSIN] |
| PRV19505 | Programa | PASO04 | Programa |
| CINTA | Fichero | PASO04 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.NEW |
| SALIDA | Fichero | PASO04 | Fichero de salidaTSDE.SALIDA.ERRORES |
| IMPRE | Fichero | PASO04 | Fichero de salidaTSDE.FICHER.PARA.IMPRI |
| PRV19505 | Programa | PASO05 | Programa |
| CINTA | Fichero | PASO05 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.NEW |

***Comprobamos DFD detallado y se ha generado correctamente:***




**El documento generado:**

SALIDA_DT_AG1JCL4_1.0.doc

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 05/06/2019 | Definición de la prueba íntegra 2h |
| | 06/06/2019 | Preparación de la prueba 3h |
| | 15/06/2019 | Finalizada la prueba |

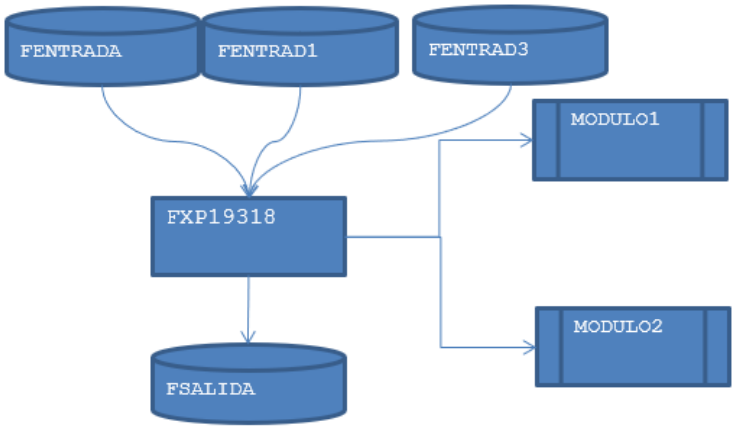
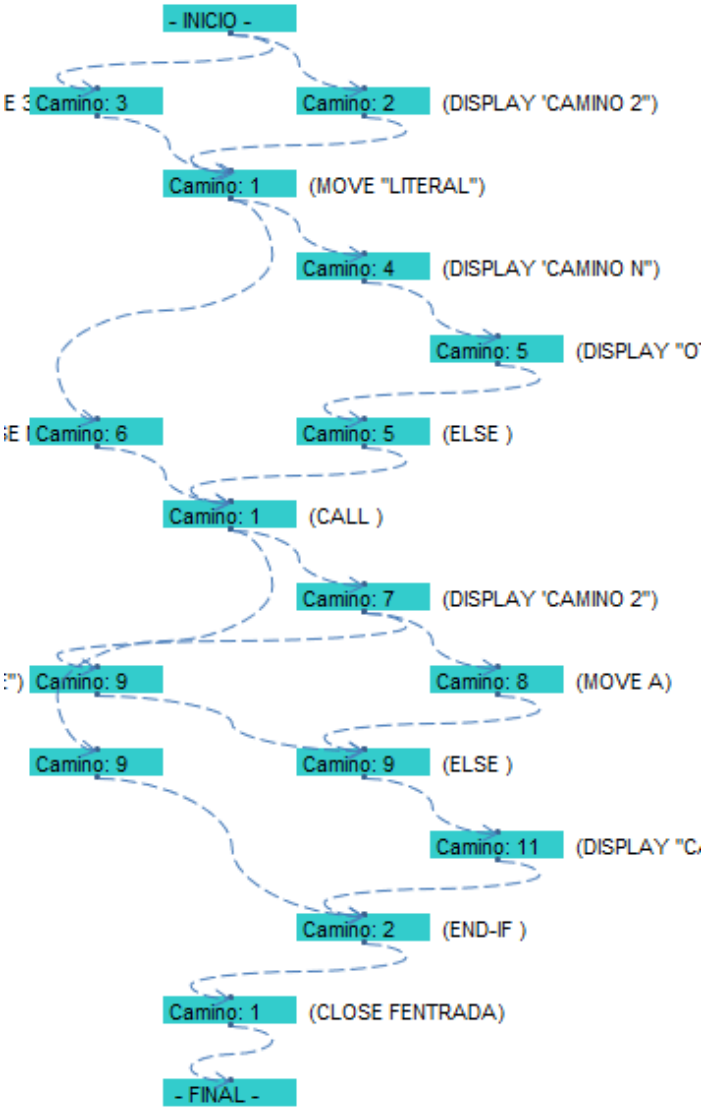
*Tabla 72. Caso 1. Evidencias de la ejecución experimental*

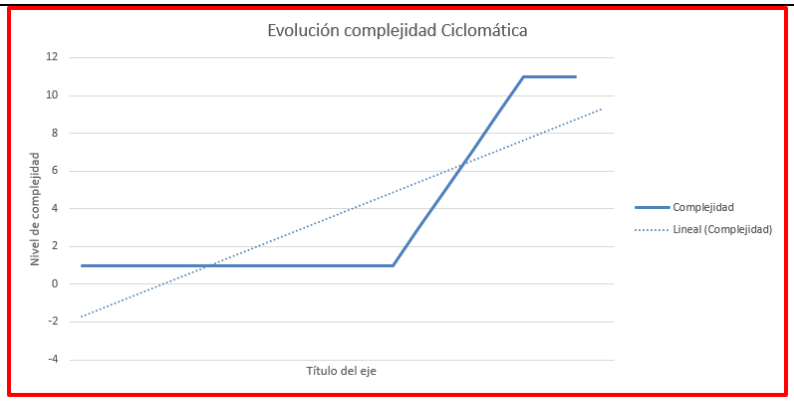
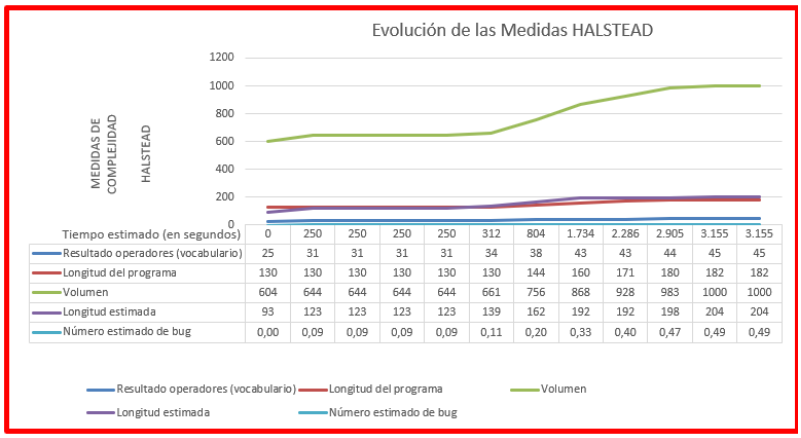
## *4.2 Caso experimental 2*

En esta ejecución, se validará el requisito REQ-002 (Generación automática de entornos de prueba Mainframes y analizador de código COBOL) y subrequisitos, junto a los requisitos no funcionales.

El código COBOL corresponderá con el programa 'ProgCobol.cob'. El resultado esperado se refleja en la tabla 73, relacionando los elementos de entrada con su correspondiente salida esperada:

| Elementos de entrada | Generación en salida |
|---|---|
| 3 ficheros de entrada.<br>1 de salida.<br>2 módulos de llamada interna. | **Diagrama de ejecución:** |

<table>
<tr>
<td></td>
<td> 
</td>
</tr>
<tr>
<td>Sentencias condicionantes que provocan 11 caminos diferentes</td>
<td>Grafo de caminos:<br> 
<br>Y complejidad ciclomática de nivel 2 (11 caminos):</td>
</tr>
</table>

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>Cálculo de las medidas Halstead a través de las variables de entrada.</td><td>Resultado de las medidas Halstead:<br> 

<table>
<tr><td>Tiempo estimado (en segundos)</td><td>0</td><td>250</td><td>250</td><td>250</td><td>250</td><td>312</td><td>804</td><td>1.734</td><td>2.286</td><td>2.905</td><td>3.155</td><td>3.155</td></tr>
<tr><td>Resultado operadores (vocabulario)</td><td>25</td><td>31</td><td>31</td><td>31</td><td>31</td><td>34</td><td>38</td><td>43</td><td>43</td><td>44</td><td>45</td><td>45</td></tr>
<tr><td>Longitud del programa</td><td>130</td><td>130</td><td>130</td><td>130</td><td>130</td><td>130</td><td>144</td><td>160</td><td>171</td><td>180</td><td>182</td><td>182</td></tr>
<tr><td>Volumen</td><td>604</td><td>644</td><td>644</td><td>644</td><td>644</td><td>661</td><td>756</td><td>868</td><td>928</td><td>983</td><td>1000</td><td>1000</td></tr>
<tr><td>Longitud estimada</td><td>93</td><td>123</td><td>123</td><td>123</td><td>123</td><td>139</td><td>162</td><td>192</td><td>192</td><td>198</td><td>204</td><td>204</td></tr>
<tr><td>Número estimado de bug</td><td>0,00</td><td>0,09</td><td>0,09</td><td>0,09</td><td>0,09</td><td>0,11</td><td>0,20</td><td>0,33</td><td>0,40</td><td>0,47</td><td>0,49</td><td>0,49</td></tr>
</table></td></tr>
<tr><td>Detección temprana de Bug (se incluyen errores leves y graves para comprobar su detección)</td><td>Deben aparecer los siguientes errores Warning:<br>Variables no utilizadas: VARIABLE2, VARIABLE3, VARIABLE4, VARIABLE5, RESULTADO, etc.<br><br>Una condición IF no cerrada<br>Deben aparecer los siguientes errores Serveros:<br>Variable de INDICE utilizada en una condición con un valor superior a su definición.</td></tr>
<tr><td>Generación del plan de pruebas unitarias</td><td>Obtención del plan de pruebas:<br>Resumen de casos de prueba:
<table>
<tr><th>ID PRUEBA</th><th>DESCRIPCIÓN</th><th>SITUACIÓN</th></tr>
<tr><td>Caso-V-1</td><td>Prueba del camino: 1 Instrucción: PROCEDURE</td><td>Pendiente</td></tr>
<tr><td>Caso-V-2</td><td>Prueba del camino: 2 Instrucción: DISPLAY 'CAMINO 2"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-3</td><td>Prueba del camino: 3 Instrucción: DISPLAY "CAMINO ELSE 3"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-4</td><td>Prueba del camino: 4 Instrucción: DISPLAY 'CAMINO N"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-5</td><td>Prueba del camino: 5 Instrucción: DISPLAY "OTRO"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-6</td><td>Prueba del camino: 6 Instrucción: DISPLAY "CAMINIO ELSE N"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-7</td><td>Prueba del camino: 7 Instrucción: DISPLAY 'CAMINO 2"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-8</td><td>Prueba del camino: 8 Instrucción: MOVE A TO B</td><td>Pendiente</td></tr>
<tr><td>Caso-V-9</td><td>Prueba del camino: 9 Instrucción: DISPLAY "PRIMER ELSE"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-10</td><td>Prueba del camino: 10 Instrucción: IF INDICE < 4</td><td>Pendiente</td></tr>
<tr><td>Caso-V-11</td><td>Prueba del camino: 11 Instrucción: DISPLAY "CAMINO ELSE 3"</td><td>Pendiente</td></tr>
<tr><td>Caso-V-12</td><td>Prueba en vacío: Realizar una prueba sin información en las entradas externas</td><td>Pendiente</td></tr>
<tr><td>Caso-V-13</td><td>Prueba estrés: Inclusión de entradas externas con mucha información</td><td>Pendiente</td></tr>
<tr><td>Caso-V-14</td><td>Pruebas búsqueda límites: Pruebas para buscar límites en bucles y condiciones</td><td>Pendiente</td></tr>
</table>
Definición de cada uno de los casos para su seguimiento:</td></tr>
</table>

ID: Caso-V-2 RESPONSABLE: Codificador

DESCRIPCIÓN: Prueba del camino: 2 Instrucción: DISPLAY 'CAMINO 2"

SITUACIÓN: Pendiente FECHA SITUACIÓN: 17/06/2019

CODIGO A PROBAR:
Instrucción: DISPLAY 'CAMINO 2"
Instrucción: ELSE
Instrucción: END-IF

DATOS ESPARADOS DE SALIDA: Evidencias de la ejecución del código especificado

**Generación del programa principal en COBOL:**

COBOL_PRINCIPAL_FXP19318.TXT: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. PRIN-FXP19318.
       AUTHOR. ABEL GARCIA.
       DATE-WRITTEN.  18/04/2019
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       FILE SECTION.
      ******************************************************************
      *      DEFINICION DE VARIABLES                                   *
      ******************************************************************
       WORKING-STORAGE SECTION.
       01 TABLA-ERRORES.
          10 W-ERROR OCCURS 100 PIC X(100).
       01 NUMERO-ERRORES        PIC 9(05).
       01 I                     PIC 9(05).
       01 MODULO                PIC X(20).

      ******************************************************************
      *      PROCESO PRINCIPAL                                         *
      ******************************************************************
       LINKAGE SECTION.
       01 RESERVA-PARM           PIC X(2).
       PROCEDURE DIVISION USING RESERVA-PARM .
       COMIENZO.
           PERFORM INICIO.
           PERFORM PROCESO.
           PERFORM FIN.

       INICIO.
      *---------*
           MOVE 0    TO NUMERO-ERRORES.
```

Línea 1, columna 1

**Y generación del entorno de ejecución:**

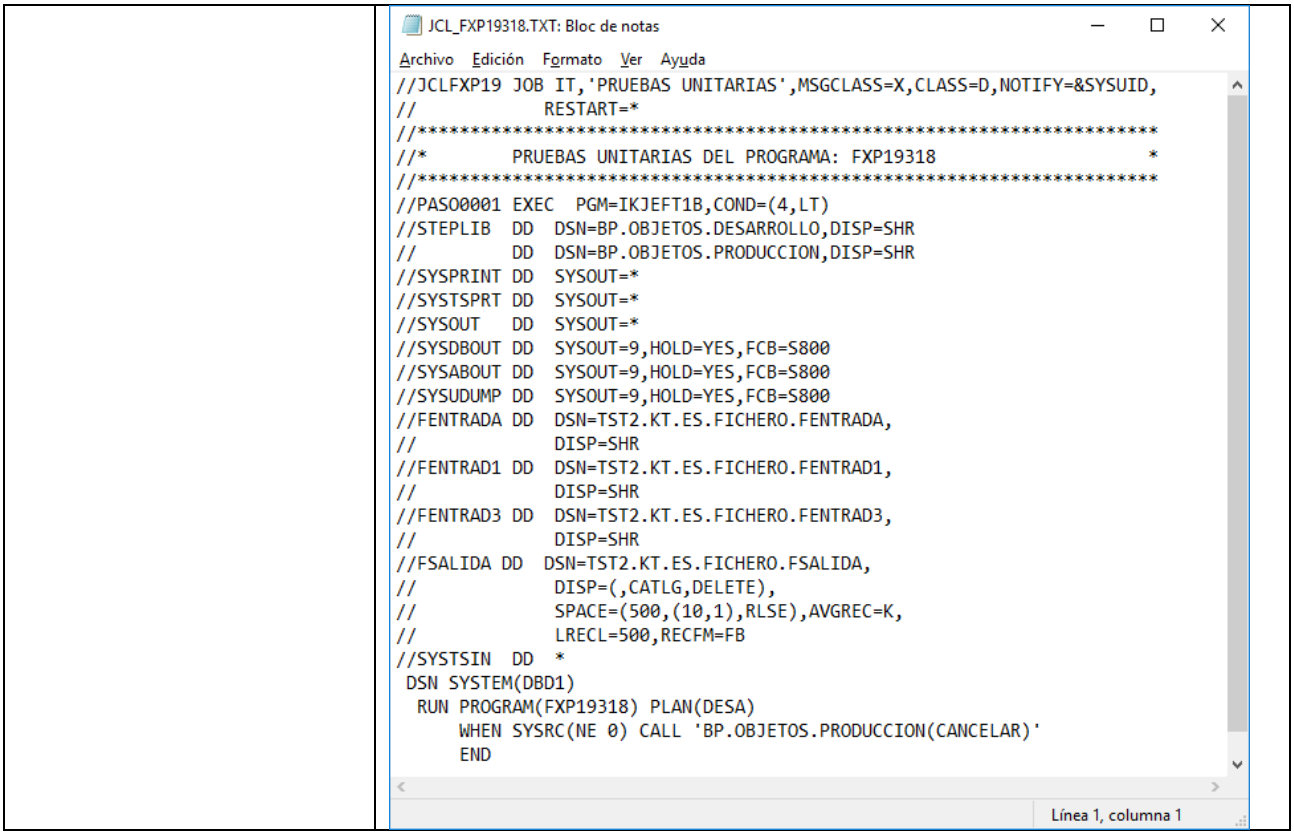



*Tabla 73. Salida esperada en el Caso Experimental 2*

En la tabla 74 se muestran las evidencias de la ejecución experimental 2 con toda la información de entrada y los resultados de salida.

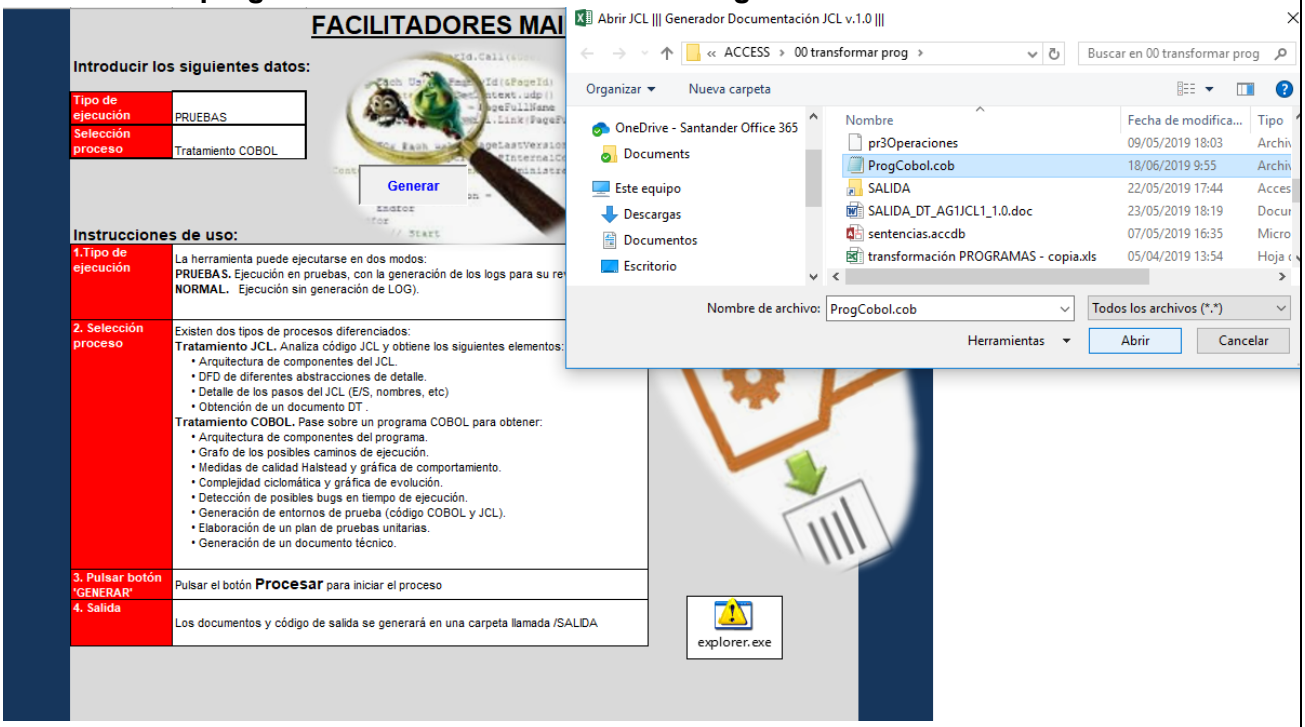
| **ID:** | **CASO-EXPERIMENTAL-002** | **Responsable:** | Abel |
|---|---|---|---|
| **Descripción** | Realizar prueba completa sobre la funcionalidad de análisis de programas COBOL y su transformación en los metamodelos definidos. | | |
| **Datos de entrada** | Código para analizar:<br>  <br>ProgCobol.cob<br><br>El programa COBOL contiene ficheros de entrada, de salida, módulos, llamadas a procesos, errores inyectados, etc. | | |
| **Datos de salida esperados** | Los datos esperados son los siguientes:<br>• Arquitectura de componentes del programa.<br>• Grafo de los posibles caminos de ejecución.<br>• Medidas de calidad Halstead y gráfica de comportamiento.<br>• Complejidad ciclomática y gráfica de evolución.<br>• Detección de posibles bugs en tiempo de ejecución.<br>• Generación de entornos de prueba (código COBOL y JCL).<br>• Elaboración de un plan de pruebas unitarias.<br>• Generación de un documento técnico. | | |

## ANÁLISIS DE LOS RESULTADOS

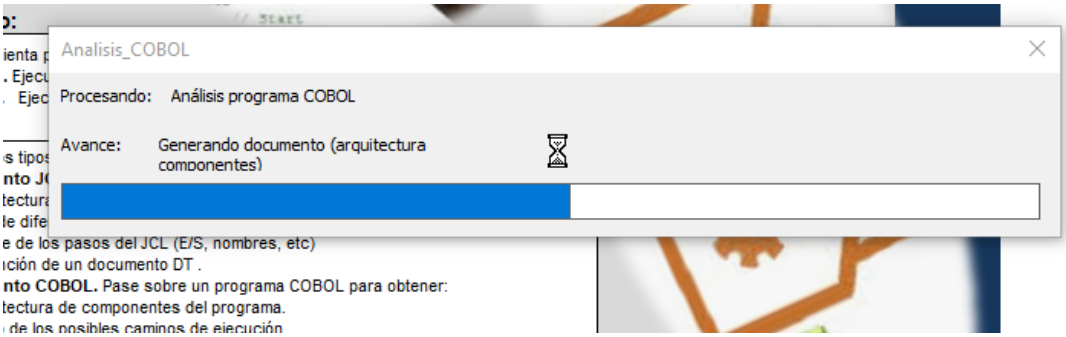
Entramos a la herra

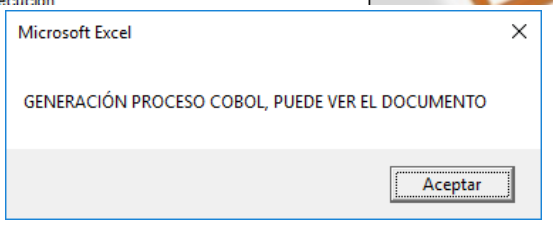
| | |
|---|---|
| **Tipo de ejecución** | NORMAL |
| **Selección proceso** | Tratamiento COBOL |

**Selección del programa COBOL a ser analizado 'ProgCobol.cob':**



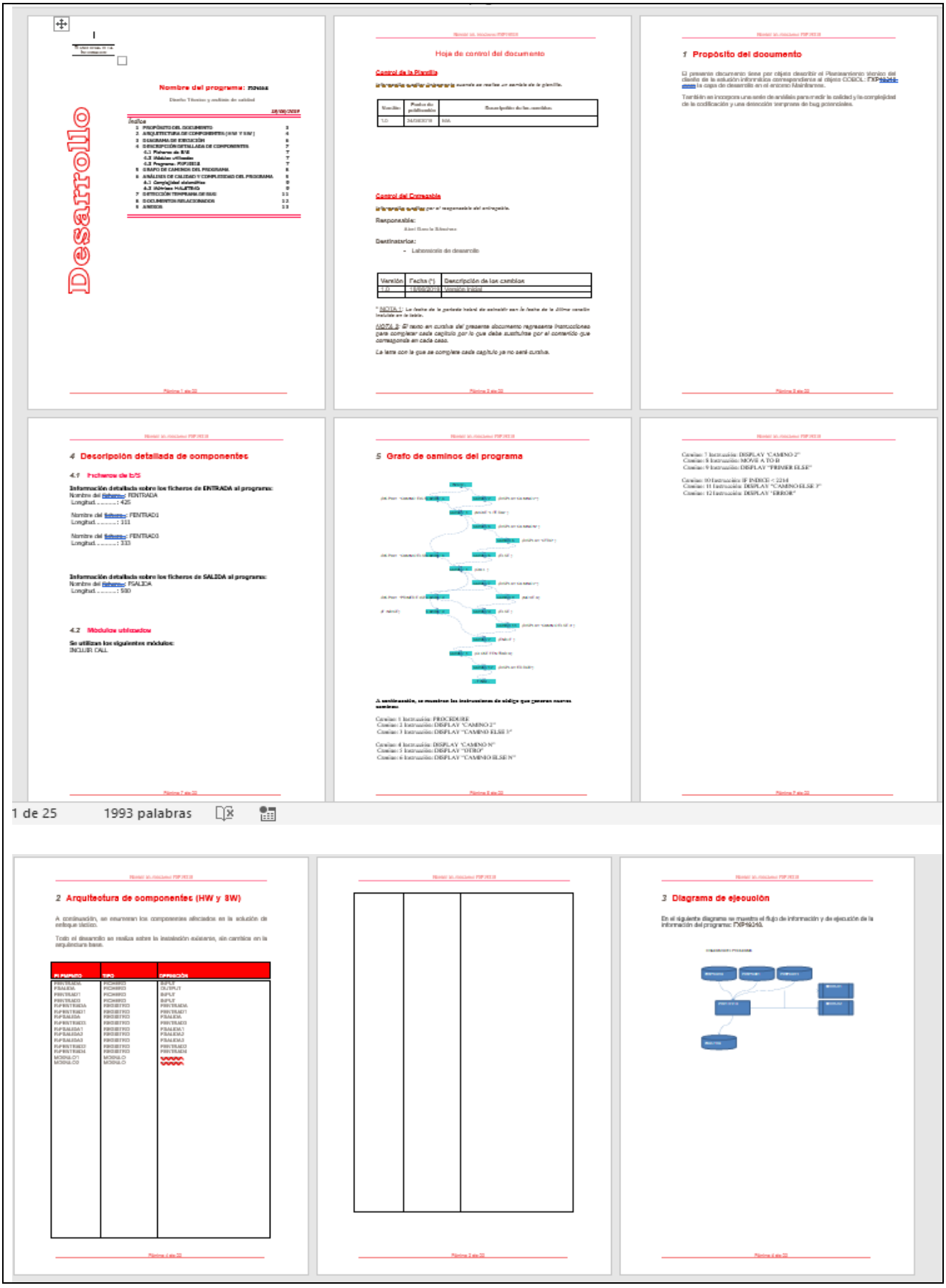

**Se inicia la generación:**

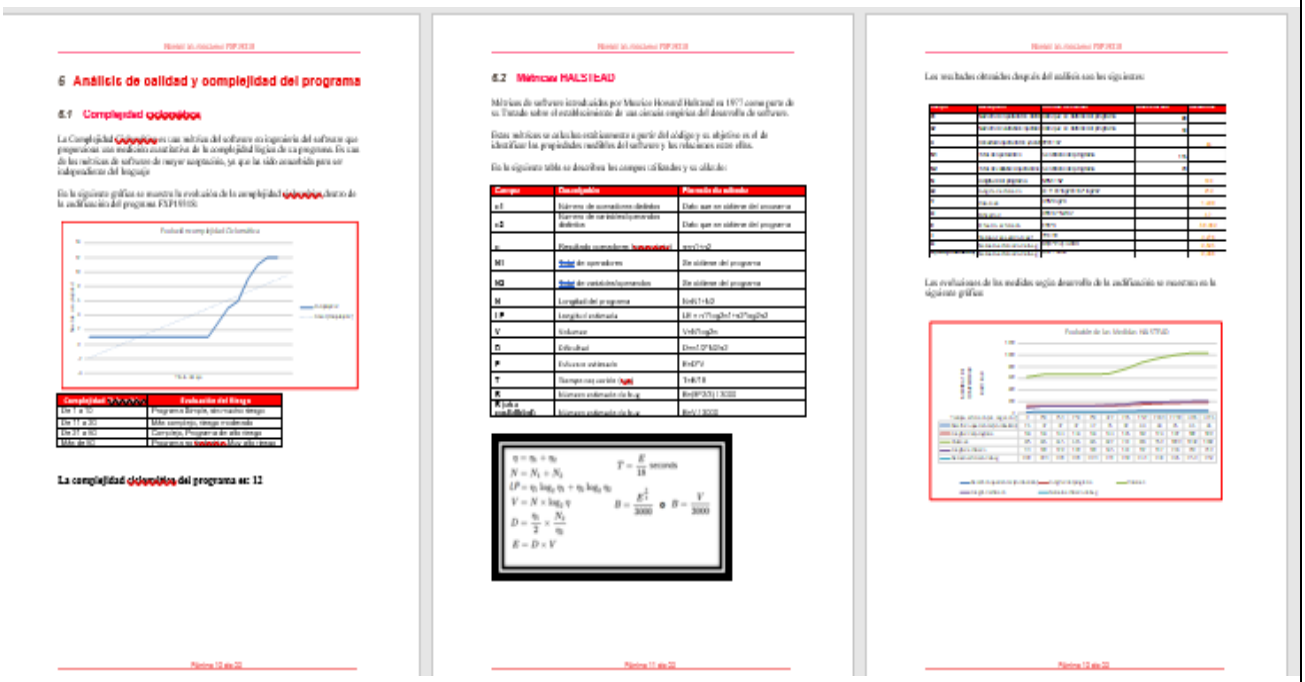


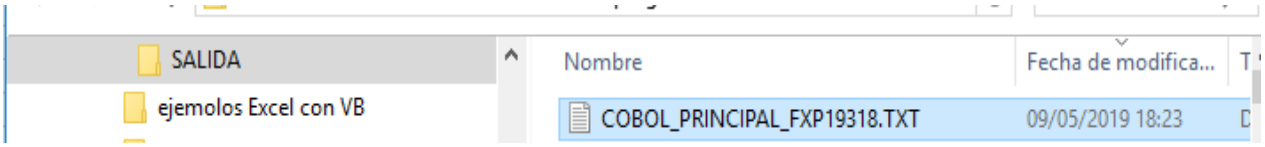




**Y finaliza OK:**

**Revisión del documento generado:**


Desarrollo
Nombre del programa: PGM0XX
Índice
Hoja de control del documento
Control de la Plantilla
Control del Entregable
1 Propósito del documento
4 Descripción detallada de componentes
4.1 Ficheros de E/S
4.2 Módulos utilizados
5 Grafo de caminos del programa
1 de 25
1993 palabras
2 Arquitectura de componentes (HW y SW)
3 Diagrama de ejecución






**Comprobación en documento generado:**

ANALISIS_FXP19318_1.0.doc

**Y revisión OK del código generado para la ejecución del entorno de pruebas (programa principal escrito en COBOL y el JCL para su lanzamiento:**

COBOL_PRINCIPAL_FXP19318.TXT

JCL_FXP19318.TXT

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 07/06/2019 | Definición de la prueba 4h |
| | 08/06/2019 | Preparación de la prueba 2h |
| | 17/06/2019 | Finalizada la prueba |

*Tabla 74. Caso 2. Evidencias de la ejecución experimental*

# CAPÍTULO 5

# 5. Conclusiones y trabajo futuro

Exposición de las conclusiones obtenidas en el desarrollo del proyecto y planteamiento de futuras líneas de trabajo relacionadas con la problemática expuesta.

## *5.1 Conclusiones*

En el presente proyecto, se ha abordado un problema complejo, al tener que generar documentos técnicos con muchas elaboraciones, transformaciones de códigos formales a estructuras visuales gráficas, obtención de medidas de complejidad, detección de errores o generación automática de otros lenguajes, para permitir realizar las pruebas unitarias en un entorno de arquitecturas basadas en mainframes y realizar analisis sobre distintas estructuras de código a través de parses.

Después de buscar varias alternativas para su solución, se opta dividir los distintos problemas y utilizar diferentes estrategias, tales como enfoques mediante paradigma de modelos (MDD y MDA), parametrización, código mixto o inyección de código, los cuales facilitan el desarrollo de proyectos al definir distintos metamodelos de entrada para ser transformados a estas estructuras a partir de los códigos formales (JCL y COBOL) y una vez generados los metamodelos, obtener todas las funcionalidades descritas en los requisitos.

Este tipo de facilitadores pueden ser muy interesantes:

- Por una parte, se pueden usar para transformar información de procesos JCL en conocimiento funcional y técnico de la aplicación, sin la necesidad de conocimiento específico de este tipo de tecnologías para el usuario (transformamos código en conocimiento).

- Y, por otra parte, el análisis del código COBOL, facilitará el desarrollo de software en Mainframes, ayudando a documentar el software, comprobando su complejidad, verificando una mínima calidad, detectando futuros errores potenciales y definiendo un entorno y un plan de pruebas unitarias.

Los beneficios económicos esperados utilizando este tipo de herramientas pueden ser muy alentadores. A continuación, se desarrollan varios casos prácticos simulados:

**Caso práctico funcionalidad GAC JCL.**

El cliente solicita al equipo de desarrollo la documentación técnica sobre todos los procesos de una sub-aplicación implantada en entorno Mainframes con una antigüedad de 10 años, de la cual no se dispone de documentación.

Requiere que diagramas de ejecución, DFD detallado y la arquitectura de componentes. La sub-aplicación cuenta con 30 procesos batch entre 10 y 30 pasos cada uno de ellos, que por método tradicional se estimaría unas 4-6 horas para documentar cada proceso JCL.

| Método | Estimación unitaria | Número de JOB | Total estimación |
|---|---|---|---|
| Tradicional | 4-6 horas | 30 | 120-180 horas |
| A través de la Herramienta | 1 hora | 30 | 30 horas |

*Tabla 75. Estimaciones de esfuerzo en documentar JOB*

Tal como se puede observar en la tabla 75, el beneficio en esfuerzo es muy elevado, además de no necesitar una persona especializada.

**Caso práctico PEP COBOL.**

En el próximo ejemplo, se desarrollará un proyecto en los que se estiman unos 50 programas de complejidad baja-media. Con herramientas de estimación del mercado, se estimaría el proyecto en unas 2.240 horas contando con todas las fases de desarrollo y documentación implícitos en éste, tal como se puede observar en la tabla 76.

| Fases del desarrollo | Estimación |
|---|---:|
| Análisis funcional | 200 |
| Diseño Técnico | 240 |
| Codificación | 840 |
| Pruebas unitarias | 328 |
| Pruebas íntegras | 440 |
| Implantación | 40 |
| Gestión y Seguimiento | 152 |
| Total Estimación | 2.240 |

*Tabla 76. Estimación desarrollo tradiciional*

Si se utilizara la herramienta desarrollada, se estima una mejora cuantitativa en todas las fases del desarrollo, pero especialmente en la fase del diseño técnico, pruebas unitarias, gestión y seguimiento:

- Diseño técnico. La herramienta genera de forma automática la documentación técnica del programa, lo que exime al programador tener que realizar el DT de unidad. Se podría cuantificar con un ahorro ente el 40% y 50% en esta fase.

- Pruebas unitarias. Se genera automáticamente el entorno de las pruebas unitarias, además de los casos de prueba y la documentación necesaria para su seguimiento. El ahorro se podría establecer entre un 20% y un 30%, además de mejorar notablemente la calidad de las pruebas (mayor mejora a menor experiencia de los codificadores).

- Gestión y seguimiento. Documentación más ordenada + código más depurado = menor número de bug. Ahorro del 5%-10%.

Según estas estimaciones, para el ejemplo, el ahorro podría suponer entre 169 y 233 horas, tal como se puede observar en la tabla 77.

| Fases del desarrollo | Estimación | % Ahorro | Horas de ahorro |
|---|---|---|---|
| Análisis funcional | 200 | 0 | |
| Diseño Técnico | 240 | 40%-50% | 96-120 |
| Codificación | 840 | 0 | |
| Pruebas unitarias | 328 | 20%-30% | 66-98 |
| Pruebas íntegras | 440 | 0 | |
| Implantación | 40 | 0 | |
| Gestión y Seguimiento | 152 | 5%-10% | 8-15 |
| **Total Estimación** | **2.240** | | **169-233 horas** |

*Tabla 77. Ahorro utilización herramienta*

A parte del beneficio económico, también existe una mejora en la confiabilidad del código, ya que la herramienta genera un juego de pruebas unitarias para realizar una depuración más exhaustiva del código.

## *5.2 Trabajo futuro*

Existen muchas posibilidades para seguir explorando y utilizando este tipo de formas de desarrollar software a través de SPL, MDD, MDA, etc. para su uso en el mundo Mainframes, dotando de nuevas funcionalidades a los facilitadores de este entorno.

En el ejemplo del proyecto desarrollado, la herramienta podría evolucionar con la incorporación de nuevas funciones que actuarían a su vez como facilitadores, aprovechando este tipo de modelos de desarrollo y que tendrían grandes sinergias con las ya actuales.

Las evoluciones más destacadas serían las siguientes:

- Arquitectura de componentes. Partiendo de los análisis de software que realiza la herramienta a través de los distintos parse y las distintas ejecuciones, se podría retroalimentar una SGBD para ir creando una arquitectura extensa sobre los componentes de la aplicación.

- Procesamiento por lotes. Mismas funcionalidades, pero realizando la ejecución de forma automática por lotes para el tratamiento de aplicaciones enteras.

- Generación automática de programas COBOL. Definición de metamodelos para la generación automática de código COBOL con distintas tipologías de programas, como por ejemplo módulos CRUD, enfrentamiento de ficheros o generación de listados entre otros.

# CAPÍTULO 6

# 6. Referencias bibliográficas


[1] *"IBM. El estándar de oro para la seguridad y la estabilidad, los Mainframes." https://www.ibm.com/es-es/it-infrastructure/servers/mainframes/mainframes. Checked: 01/06/2019.*

[2] *Alberto Iglesias Fraga, "El mainframe sigue muy vivo", revista Ticbeat, 19/09/2019.*

[3] *Oscar Humberto, "Dónde estarán los mainframes en diez años", revista TechData, 05/04/2018.*

[4] *"IBM. Definición y evolución de los Mainframes". https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_intro.html Checked: 01/06/2019.*

[5] *Rubén Heradio Gil, "Metodología de desarrollo de software basada en el paradigma generativo. Realización mediante la transformación de ejemplares". Ph. D. Thesis, UNED, España, abril 2007.*

[6] *Jack Herrington, "Code Generation in Action", 2003. MANNING ISBN-13: 9-781930110977.*

[7] *Eric J. Evans, "Domain-Driven Design: Definitions and Pattern Summaries", Dog Ear Publishing, LLC, 22/09/2014, ISBN-13: 978-1457501197*

[8] *Martin Fowler, Rebecca Parsons, 'Domain-Specific Languages', 2010, Addison-Wesley Educational Publishers Inc, ISBN-13: 978-0321712943*

[9] *"Productos relacionados con el lenguaje de programación COBOL" https://www.microfocus.com/es-es/products/cobol-development/, Checked: 01/06/2019.*

[10] *\Microfocus, www.**microfocus**.com/es, Checked: 01/06/2019.*

[11] *\VB, https://docs.microsoft.com/es-es/dotnet/visual-basic/ Checked: 01/06/2019.*

[12] \VB, https://es.wikipedia.org/wiki/Visual_Basic. Checked: 01/06/2019.

[13] IBM, Manual del código JCL, https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.ieab600/toc.htm Checked: 01/06/2019.

[14] \Genexus, https://www.genexus.com/ Checked: 01/06/2019.

[15] \Genexus, https://www.lawebdelprogramador.com/foros/GeneXus/32835-QUE-ES-GENEXUS-PARA-QUE-SIRVE.html. Checked: 01/06/2019.

[16] Eduardo Vélez Aguirre, Moisés López Bermúez, "Análisis de la obsolescencia de software por el avance de la tecnología", 2019, 201.159.222.36

[17] \CODEDOM, https://msdn.microsoft.com. Checked: 01/06/2019.

[18] \Gvnix, www.gvnix.org. Checked: 01/06/2019.

[19] Enrique Ruiz Valenciano, Mario Martínez Sánchez, "Desarrollo rápido de geoportales con gvNIX ",ISSN 1131-9100, Nº 169, 2015, pp. 32-35

[20] \Generadores de código POJO, https://www.ibm.com/support/knowledgecenter/es/SSQP76_8.0.1/com.ibm.wodm.dserver.rules.res.managing/topics/tpc_res_overview_code_generators.html. Checked: 01/06/2019.

[21] Iwan Binanto, Harco Leslie Hendric Spits Warnars, Bahtiar Saleh Abbas, Nesti Fronika Sianipar, "Automation Processing Halstead Metrics Application's Results",7-8 septiembre 2018, Indonesian Association for Pattern Recognition International Conference (INAPR), Jakarta.

[22] Warren Harrison, Kenneth Magel, Raymond y Arlan DeKock, "Applying Software Complexity Metrics to Program Maintenance", Missouri University of Science and Technology, 1982, pp 68-60.

[23] Roger S. Pressman, "Ingeniería del Software, un enfoque práctico", Mc Graw Hill 5ª edición. ISBN: 84-481-3214-9.

[24] Thomas J. McCabe, "A complexity measure", IEEE Transactions on Software Engineering, Vol. SE-2, December 1976, pp.43-55.

[25] \Herramientas de calidad de software, https://www.javiergarzas.com/2012/03/herramientas-de-calidad-software.html.. Checked: 01/06/2019.

[26] \SONAR, https://www.sonarsource.com/products/codeanalyzers/sonarcobol.html Checked: 01/06/2019.

[27] Andreas Spillner, Tilo Linz, Hans Schaefer,"Software Testing Foundations: A Study Guide for the Certified Tester Exam (Rocky Nook Computing)", Rocky Nook, 4ª

*Edición (29 de marzo de 2014),ISBN-13: 978-1937538422*

*[28] José de Jesús Jiménez Puello, Tomás San Feliu, Jose Antonio Calvo-Manzan, "Una aproximación basada en metamodelado del área de proceso de Validación del CMMI: Un caso de estudio". Revista RISTI, 2016, pp. 45-58.*

*[29] Jose Luis Aristegui O., "Los casos de prueba en la prueba de software". Revista Digital Lámpsakos, No. 3, 2010, pp. 27-34.*

*[30] \JUNIT." www.junit.org, Checked: 01/06/2019.*

*[31] Danay Larrosa-Uribazo, Perla Beatriz Fernandez and Martha Dunia Delgado-Dapena, "GeCaP: Generador de casos de pruebas unitarias a partir del código fuente en lenguaje Java", vol. 57, 2018, ISSN 2395-8618, pp. 67–73.*

*[32] \NUNIT, www.nunit.org.. Checked: 01/06/2019.*

*[33] \Xpediter/CICSCOBOL User Guide Release 17.02 http://frontline.compuware.com/Doc/XD/XD1702/PDF/XpediterCICS_COBOLUserGuide.pdf Checked: 01/06/2019.*

*[34] Paulo J. Pires y Fernando Brito e Abreu, "Knowledge Discovery Metamodel-based Unit Test Cases Generation", 2018 IEEE 11th International Conference on Software Testing, Verification and Validation, ISTAR-IUL Lisboa, Portugal*

*[35] Roy Patrick Tan and Stephen Edwards, "Evaluating Automated Unit Testing in Sulu", Department of Computer Science 660 McBryde Hall (0106), 2008, Blacksburg, Virginia.*

*[36] Ian Sommerville, "Ingeniería del Software", (verificación y validación), Addison Wesley, edición 2011, ISBN 9786073206037.*

*[37] \Ranking estadísticos TIOBE, https://www.tiobe.com, Checked: 01/06/2019.*

*[38] \Ranking de lenguajes de programación, https://www.digitallearning.es/blog/rankings-de-lenguajes-de-programacion/, Checked: 01/06/2019.*

*[39] \Análisis de ranking, https://www.tiobe.com/tiobe-index/, Checked: 01/06/2019.*

*[40] Hanna, M., "Farewell to Waterfallm", Software Magazine, Mayo 1995, pp.38-46.*

*[41] Jean-Paul Subra, Aurélien Vannieuwenhuyze,"Scrum. Un método ágil para sus proyectos", ENI (26 de febrero de 2018), ISBN-13: 978-2409012921.*

*[42] Kaushal Chari, Manish Agrawal,"Impact of incorrect and new requirements on waterfall software project outcomes", Article 2018, Volume 23, Issue 1, pp 165–185.*

*[43] Gonzalo Cuevas Agustín, "Gestión del proceso software (Técnicas de gestión de proyectos)", 2005, Centro de estudios Ramón Areces S.A. ISBN: 9788480045469.*

*[44] Mario Piattini Velthuis, Jose Antonio Calvo Manzano, Jose Cervera Bravo, "Análisis y diseño de aplicaciones informáticas de gestión, una perspectiva de ingeniería del software". México: Alfaomega Grupo Editor, 2004, pp.419-469.*

*[45] Patricia Mouy, Bruno Marre, Nicky Williams, "Generation of all-paths unit test with function calls", CEA/LIST, LSL, 91191 Gif Sur Yvette Cx, 2008, Paris .(2008 International Conference on Software Testing, Verification, and Validation).*

*[46] CMMI Institute, CMMI® para Servicios, Versión 1.3, http://cmmiinstitute.com/ pp. 483-500.*

*[47] Arloys Macías-Rojas, Martha Dunia Delgado-Dapena, Jenny Fajardo-Calderín and Danay Larrosa-Uribazo, "Generador de valores de casos de pruebas funcionales". Lámpsakos, No.15, pp. 51-58, 2016, ISSN: 2145-4086, Medellín–Colombia https://dialnet.unirioja.es/descarga/articulo/5403332.pdf,*

*[48] Jaroslaw Berlowski, Patryk Chrusciel, Marcin Kasprzyk, Iwona Konaniec, Marian Jureczko, "Highly Automated Agile Testing Process: An Industrial Case Study", e-Informatica Software Engineering Journal. 2016;10(1):69-87 DOI 10.5277/e-Inf160104.*

*[49] Tom Simonite, "That Handle Our Most Sensitive Data Are Open to Internet Attacks", mit the technology review, 5 agosto 2015.*

# ANEXOS

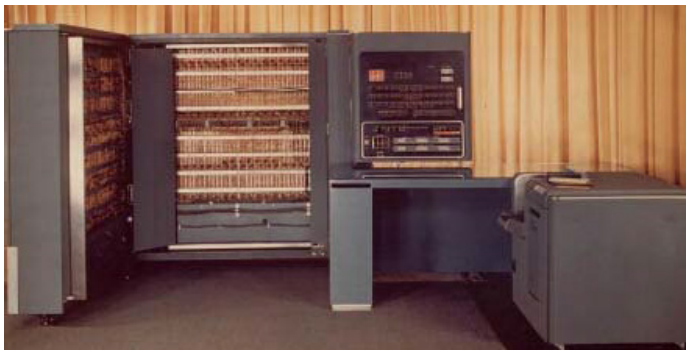
## *Acrónimos*

| Acrónimo | Descripción |
|---|---|
| B2B | business to business |
| B2C | bussiness to consumer |
| CodeDOM | Code Document Object Model |
| XLST | EXtensible Stylesheet Language |
| COBOL | COmmon Business-Oriented Language |
| MDD | *Model-driven development* |
| MDA | Model-Driven Architecture |
| SPL | Software Product Line |
| API | Application Programming Interface |
| UNED | Universidad Nacional de Educación a Distancia. |
| SOA | Software Orientado a Arquitecturas |
| JCL | Job Control Lenguage |

## *Evolución de sistemas Mainframes IBM*

A continuación, en la tabla 76, se hace una breve reseña sobre la evolución de estos equipos, donde la compañía de IBM es la gran precursora y la compañía que tiene la mayor cuota de mercado en estos sistemas:

| Equipo | Descripción |
|---|---|
| IBM 701 | • Lanzada al mercado en 1952 (primera generación), fue la primera computadora electrónica a gran escala fabricada por IBM y su primera computadora científica en el mercado.<br>• Primera máquina de IBM en la que los programas se almacenaron en una interna.<br>• Y la primera computadora de la serie 700 que iría desde la 701 a 709. |

| | |
|---|---|
| |  <br>*Figura 34. Computadora IBM 701* |
| IBM 305 RAMAC | • 1956 (primera generación), primera computadora en incluir unidad de disco.<br>• La capacidad de todo el archivo de disco fue de 5 millones de caracteres de 7 bits. |
| IBM 1401 | • 1959 (segunda generación). Su sistema de procesamiento de datos ya era totalmente transistorizado.<br>• Comercialización para pequeñas empresas.<br>• Uso tarjeta perforada convencional.<br>• Utilización de cinta magnética de alta velocidad (para su tiempo)<br>• Impresión.<br>• Capacidad de tener programas almacenados. |
| IBM 7094 | • 1962 (segunda generación). Computación científica a gran escala.<br>• Gran aumento de velocidad de procesamiento en comparación con sus antecedores.<br>• Inicio del uso de coma flotante de doble precisión. |
| Sistemas 360 | • 1964. Inicio de la tercera generación.<br>• Nueva arquitectura, diseñado tanto para el procesamiento de datos y para ser compatible a través de una amplia gama de niveles de rendimiento.<br>Hardware de la computadora:<br>▪ Direccionamiento de 24 bits (arquitectura de 32 bits).<br>▪ Almacenamiento principal, el tamaño máximo es de 16MB<br>▪ Una o dos unidades centrales de procesamiento (CPU)<br>▪ Uno a siete canales.<br>▪ Selector o Multiplexor Byte<br>▪ Multiplexor de bloque<br>▪ Unidades de control (que se conectan a los canales)<br>▪ Dispositivos (que se conectan a las unidades de control) |

| | |
|---|---|
| | Los Sistemas operativos utilizados:<br>▪ Familia de sistemas operativos de IBM.<br>▪ Sistema operativo / 360 (OS / 360)<br>▪ Sistema operativo de disco / 360 (DOS / 360)<br>▪ Sistema Operativo de Cinta (TOS)<br>▪ Soporte de programación básica (BPS)<br>▪ Programa de Control de Aerolíneas |
| IBM 9020 | • El IBM 9020 fue creado a partir de la unión de computadoras IBM System 360, en concreto del modelo 65s.<br>• Utilizado para el control aéreo. Ejemplo del panel de control:<br>  <br>*Figura 35. Panel de control 9020* |
| Sistemas 370 | • 1970, compatibles con los sistemas 360.<br>• El modelo 145 de la 370 es la primera computadora con monolítico totalmente integrado.<br>• Nuevos periféricos. Disco 3330/3340/3350 y 3211 impresora<br>• Primer multiprocesador modelos 158MP y 168MP<br>• Evolución de los sistemas operativos de IBM:<br>▪ OS/360 ➔ OS/VS<br>▪ DOS/360 ➔ DOS/VS<br>▪ CP/67 ➔ VM/370<br>• Almacenamiento virtual, direccionamiento, etc. |

| | |
|---|---|
| | • **Arquitectura de** 256 canales<br>• **CPU changes** Extended MP support via CPU Address<br>  <br>*Figura 36. Sistemas 370* |
| Sistemas 390 | • 1990, evolución de ESA/370.<br>• 1994 – S/390 Parallel Transaction Server<br>– Familia de procesadores CMOS.<br>• 1998 – System/390 Generation 5 server – mas de 1.000 MIPS<br>• 1999 – System/390 Generation 6 server – copper chip technology<br>• Familia de SO:<br>– MVS/ESA ➔ OS/390<br>– VSE/ESA<br>– VM/ESA<br>– AIX/ESA<br>– Linux for S/390 (Diciembre de 1999) |
| Z Series | • Año 2000. Evolución ESA-390.<br>• Direccionamiento soportado de 24-bit, 31-bit y 64-bit.<br>• Procesadores:<br>– z900 – up to 16 processors<br>– z800 – up to 4 processors<br>• Familia de SO<br>– OS/390 ➔ z/OS<br>– VSE/ESA ➔ z/VSE<br>– VM/ESA ➔ z/VM<br>– TPF ➔ z/TPF<br>– Linux for S/390 ➔ Linux for zSeries |

| | |
|---|---|
| |  <br>*Figura 37. Serie Z*<br><br>Los últimos sistemas Z de IBM, han fortalecido el papel del mainframe dentro de la empresa, gracias a su capacidad de adaptación, gran rendimiento e incorporación de nuevas tecnologías.<br><br>Características de los sistemas Z:<br><br>• Gran capacidad de virtualización.<br>• Mejoras en la conexión a los datos y la red pueden ayudar a proporcionar acceso más rápido a los datos.<br>• Despliegue Just-In-Time de recursos.<br>• Los motores especializados ofrecen una alternativa atractiva cuando se ejecutan nuevos.<br>• Capacidad de ampliación a más de 50 Mil millones de instrucciones por segundo (BIPS) |

Tabla 78. Evolución sistemas Mainframes de IBM

Aunque algunos artículos pongan en duda la seguridad en los Mainframes [49], se ha demostrado que estos sistemas con configuraciones consolidadas en grandes compañías no muestran tal debilidad.

***Planificación proyecto***

## 1. Objetivo del proyecto y situación general del avance

| | |
|---|---|
| **Objetivo del proyecto** | **Generación automática de código, documentación y datos de prueba unitarias para entornos productivos Mainframes** |
| **Alcance** | Los requerimientos detectados para el desarrollo son los siguientes:<br>• Generación automática de documentación técnica sobre procesos JCL.<br>• Generación automática de entornos de prueba Mainframes y analizador de código COBOL<br>• Otros requisitos NO FUNCIONALES<br>• Documentación asociada al proyecto |
| **Enfoque** | Utilización metodología clásica |

| Situación general | ~~• Estado inicial del proyecto con propuesta inicial.~~<br>~~• Pendiente GO.~~<br>• GO el 26-02-2019<br>• Inicio codificación primer requisito |
|---|---|

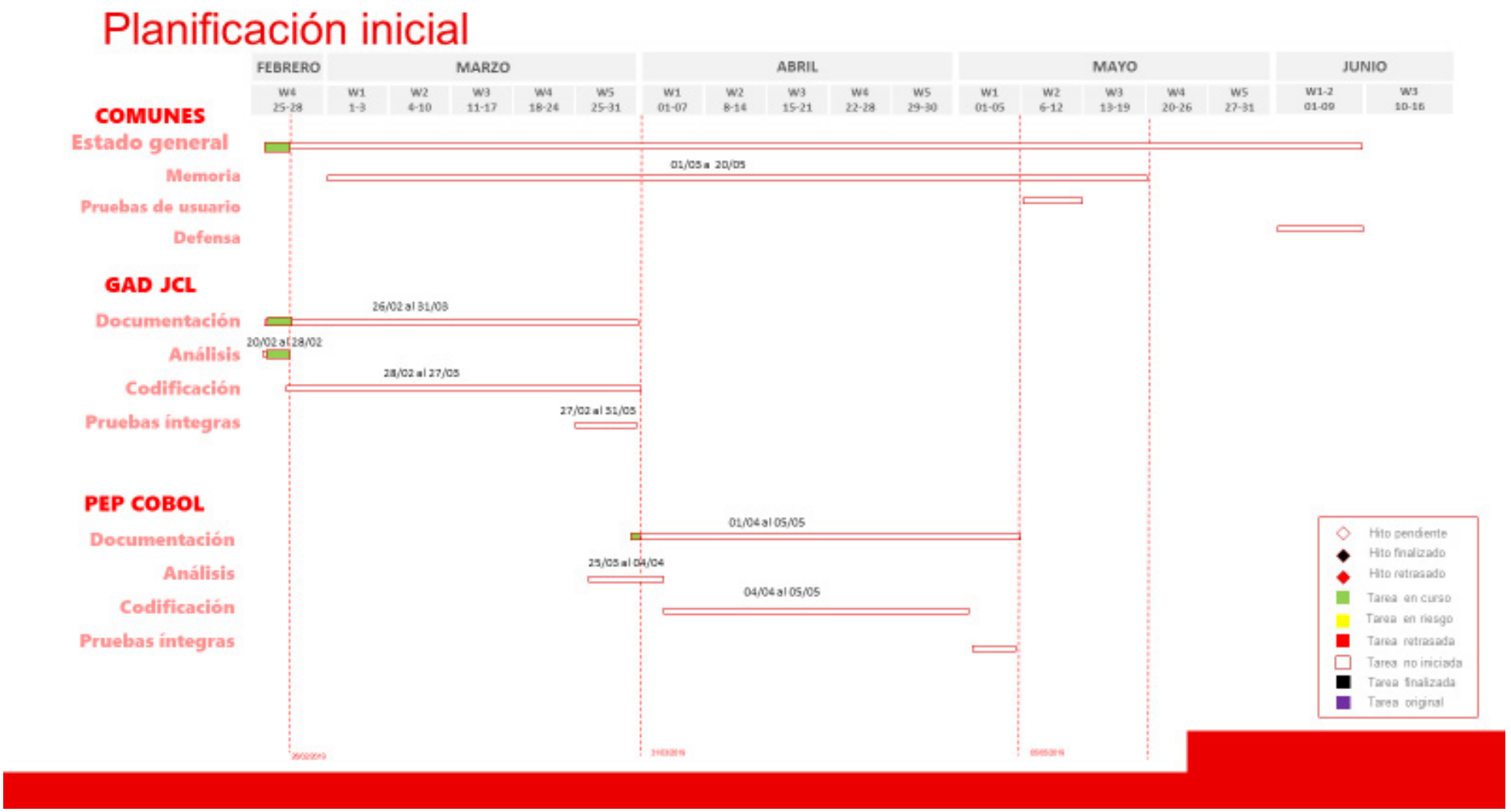
## 2. Revisión de Tareas/hitos

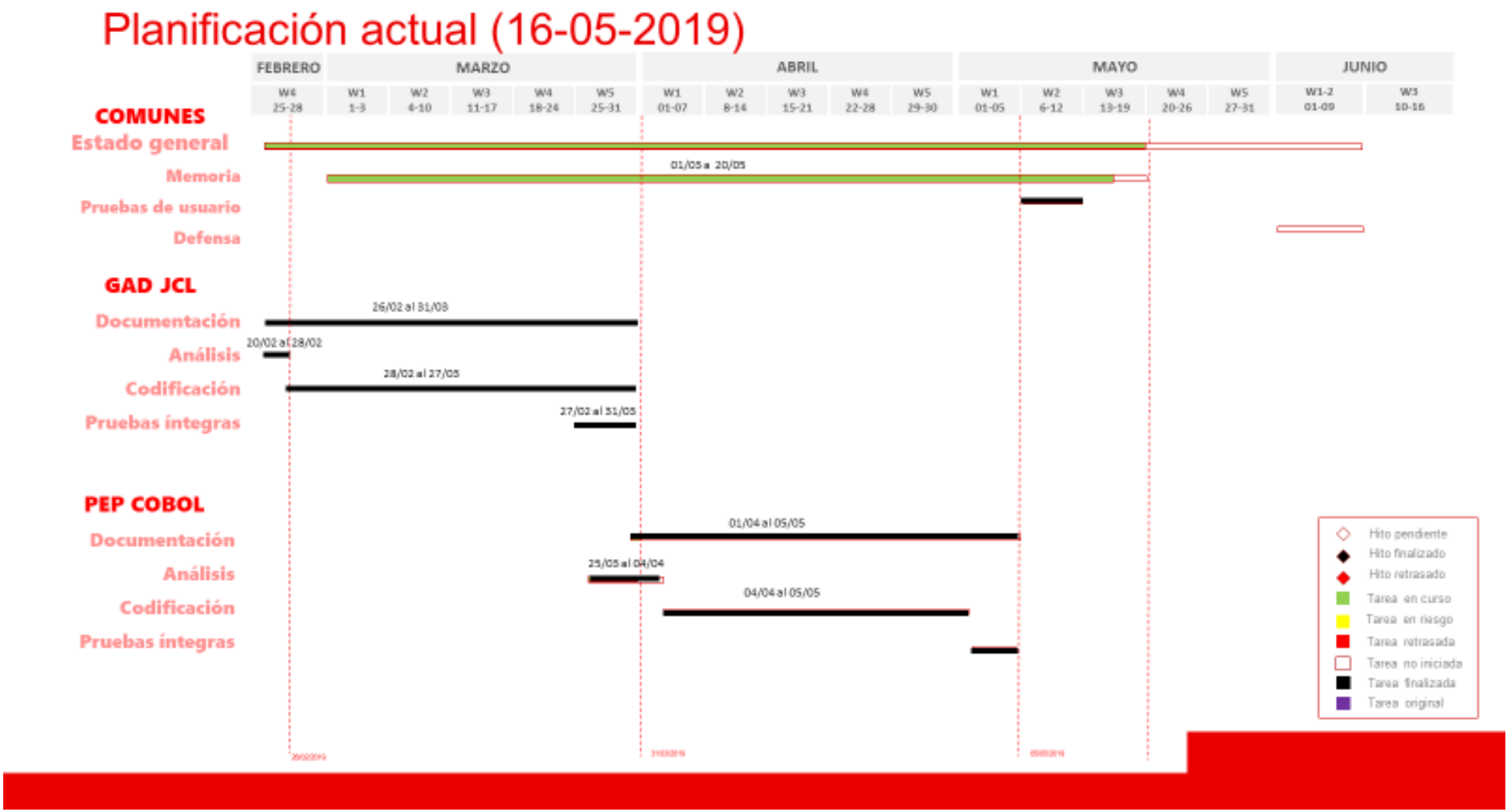
| Actividades realizadas / en curso (a fecha 01-03-2019) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Descripción | Fecha Real | | Fecha Prevista | | Esfuerzo | % Avance | % Avance Esperado | Situación | Comentarios |
| | Inicio | Fin | Inicio | Fin | | | | | |
| Lectura de artículos e investigación sobre temas a tratar en el proyecto | 15/01/19 | 05/02/19 | 15/01/19 | 05/02/19 | 40 | 100% | 100% | ☑ | Finalizado OK |
| Realización de propuestas, | 17/01/19 | 26/02/19 | 17/01/19 | 20/02/19 | 10 | 100% | 100% | ☑ | Finalizado con pequeño retraso |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| reuniones con el tutor y selección de los contenidos finales del proyecto | | | | | | | | | |
| Documentación (análisis, dfd, arquitecturas planteadas, etc) | 26/02/19 | 30/03/19 | 26/02/19 | 31/03/19 | 25 | 10% | 10% | ● | Iniciado |
| Definición de estructuras de datos y metamodelos. | 26/02/19 | 26/02/19 | 26/02/19 | 27/02/19 | 10 | 100% | 100% | ☑ | Finalizado OK |
| Codificación y pruebas unitarias del parse para JCL (obtención de objetos) | 26/02/19 | 01/03/19 | 28/02/19 | 03/03/19 | 35 | 50% | 50% | ● | Iniciado |
| Codificación y pruebas unitarias generación DFD nivel 1 | 01/03/19 | 03/03/19 | 04/03/19 | 07/03/19 | 15 | 0% | 0% | ● | Pendiente inicio |
| Codificación y pruebas unitarias generación DFD nivel 2 | 04/03/19 | 25/03/19 | 08/02/19 | 22/03/19 | 50 | 0% | 0% | ● | Pendiente inicio |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Codificación y pruebas unitarias de la generación del documento técnico | 26/03/19 | 28/03/19 | 23/03/19 | 27/03/19 | 20 | 0% | 0% | ● | Pendiente inicio |
| Preparación de casos de prueba y documentación de éstas. | 28/03/19 | 30/03/19 | 28/03/19 | 31/03/19 | 15 | 0% | 0% | ● | Pendiente inicio |
| Documentación (análisis, dfd, arquitecturas plan, pruebas, etc) | | | 01/04/19 | 05/05/19 | 25 | 0% | 0% | ● | Pendiente inicio |
| Definición de estructuras de datos y metamodelos. | | | 01/04/19 | 03/04/19 | 10 | 0% | 0% | ● | Pendiente inicio |
| Codificación del Parse lenguaje COBOL | | | 04/04/19 | 13/04/19 | 35 | 0% | 0% | ● | Pendiente inicio |
| Generación automática de código (programa COBOL y JCl) para entorno de pruebas | | | 14/04/19 | 18/04/19 | 20 | 0% | 0% | ● | Pendiente inicio |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Estimar calidad y complejidad por métodos empíricos | | | 19/04/19 | 03/05/19 | 60 | 0% | 0% | ● | Pendiente inicio |
| Refundir toda la información en un documento DT y pruebas íntegras. | | | 03/05/19 | 05/05/19 | 15 | 0% | 0% | ● | Pendiente inicio |
| Pruebas íntegras de toda la herramienta y correcciones | | | 05/05/19 | 10/05/19 | 20 | 0% | 0% | ● | Pendiente inicio |
| Documentación de la memoria | | | 01/03/19 | 20/05/19 | 50 | 20% | 20% | ● | Desarrollo en paralelo de la memoria, cumplimentando los apartados técnicos del desarrollo y otros apartados de análisis e investigación. |
| Planificación y seguimiento del proyecto | 26/02/19 | | 26/02/19 | 15/06/19 | 10 | 30% | 30% | ● | Seguimiento según previsto. |
| Preparación de la defensa | 01/06/19 | | 01/06/19 | 15/06/19 | 20 | 0% | 0% | ● | Pendiente inicio |


Planificación inicial
FEBRERO
MARZO
ABRIL
MAYO
JUNIO
W4 25-28
W1 1-3
W2 4-10
W3 11-17
W4 18-24
W5 25-31
W1 01-07
W2 8-14
W3 15-21
W4 22-28
W5 29-30
W1 01-05
W2 6-12
W3 13-19
W4 20-26
W5 27-31
W1-2 01-09
W3 10-16
COMUNES
Estado general
Memoria
Pruebas de usuario
Defensa
01/05 a 20/05
GAD JCL
Documentación
Análisis
Codificación
Pruebas íntegras
26/02 al 31/03
20/02 al 28/02
28/02 al 27/05
27/02 al 31/05
PEP COBOL
Documentación
Análisis
Codificación
Pruebas íntegras
01/04 al 05/05
25/03 al 04/04
04/04 al 05/05
Hito pendiente
Hito finalizado
Hito retrasado
Tarea en curso
Tarea en riesgo
Tarea retrasada
Tarea no iniciada
Tarea finalizada
Tarea original


Planificación actual (16-05-2019)
FEBRERO
MARZO
ABRIL
MAYO
JUNIO
W4 25-28
W1 1-3
W2 4-10
W3 11-17
W4 18-24
W5 25-31
W1 01-07
W2 8-14
W3 15-21
W4 22-28
W5 29-30
W1 01-05
W2 6-12
W3 13-19
W4 20-26
W5 27-31
W1-2 01-09
W3 10-16
COMUNES
Estado general
Memoria
Pruebas de usuario
Defensa
01/05 a 20/05
GAD JCL
Documentación
Análisis
Codificación
Pruebas íntegras
26/02 al 31/03
20/02 al 28/02
28/02 al 27/05
27/02 al 31/05
PEP COBOL
Documentación
Análisis
Codificación
Pruebas íntegras
01/04 al 05/05
25/03 al 04/04
04/04 al 05/05
Hito pendiente
Hito finalizado
Hito retrasado
Tarea en curso
Tarea en riesgo
Tarea retrasada
Tarea no iniciada
Tarea finalizada
Tarea original

## 3. Situación de entregables

| Hito | Entregables | Responsable / Implicado | F. Inicio | F. fin | Situación | |
|---|---|---|---|---|---|---|
| **Fase 1** | Propuestas iniciales | Abel | 10/01/19 | 21/01/19 | Entregadas y con reporte | ☑ |
| | Análisis e investigación de propuestas | Abel | 21/01/19 | 01/02/19 | Entregadas y con reporte | ☑ |
| | Propuesta definitiva | Abel | 10/02/19 | 15/02/19 | Entregadas y con reporte | ☑ |
| | Documento VB inicio proyecto | Ruben | 20/02/19 | 26/02/19 | Entregadas y con reporte | ☑ |
| **Fase 2** | Memoria con la información correspondiente a la parte de la funcionalidad de documentación | Abel | 26/02/19 | 31/03/19 | Iniciado | ● |
| | Herramienta con la funcionalidad de documentación | Abel | 26/02/19 | 31/03/19 | Iniciado | ● |
| **Fase 3** | Ajustes fase2 | Abel | 01/04/19 | 10/04/19 | Pendiente inicio | ● |
| | Memoria incluyendo información de pruebas | Abel | 01/04/19 | 05/05/19 | Pendiente inicio | ● |
| | Herramienta con la funcionalidad de pruebas unitarias | Abel | 16/03/19 | 05/05/19 | Pendiente inicio | ● |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Fase 4** | Ajustes fase3 | Abel | 05/05/19 | 20/05/19 | Pendiente inicio | ● |
| | Entrega de la memoria completa | Abel | 05/05/19 | 20/05/19 | Pendiente inicio | ● |
| | Entrega de la herramienta completa | Abel | 05/05/19 | 20/05/19 | Pendiente inicio | ● |
| **Entrega final** | Ajustes de la entrega | Abel | 16/05/19 | 01/06/19 | Pendiente inicio | ● |
| | Preparación de presentación | Abel | 16/05/19 | 01/06/19 | Pendiente inicio | ● |

## 4. Riesgos detectados

| Id. | Criticidad | Descripción | Acción mitigadora |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1 | Alta | Pendiente GO<br>GO al proyecto 26-02-2019 | • Propuesta temprana de iniciativas para poder disponer de tiempo de reacción ante cambios no contemplados.<br>• Solicitud documentación con VB.<br>• Permitir ajustes en la propuesta inicial. |
| 2 | Media | Complejidad en la transformación de código a entorno visual (creación DFD para documentos técnicos). | • Investigar y analizar herramientas disponibles para crear formas gráficas.<br>• Crear prototipos de pruebas que se puedan reutilizar en la herramienta.<br>• Consultas foros. |
| 3 | Media | Complejidad en la creación automática de pruebas unitarias en programas COBOL, | • Investigar y analizar herramientas disponibles.<br>• Analizar métodos de inspección de código.<br>• Crear prototipos sobre algoritmos de inspección.<br>• Consultas foros. |
| 4 | Media | Desvío en los plazos u objetivos del desarrollo. | • Revisión periódica de la evolución del proyecto.<br>• Realización de entregas parciales en formato MPV (mínimo producto viable), con feedback del cliente, |

## *Evidencias ejecución de pruebas*

Pruebas de validación de sub requisitos

| ID: | CASO-U-001 | Responsable: | Abel |
|---|---|---|---|
| Funcionalidad/ Requisitos | **Validación del sub requisito REQ-001-01** Analizador semántico de JCL para obtener la arquitectura de componentes software del JCL | | |
| Descripción | Verificación de la correcta extracción de los elementos software de un proceso JCL. | | |
| Datos de entrada | Selección del siguiente proceso JCL con 2 pasos: | | |

```
//AG1SORT  JOB IT,DESARROLLO,MSGCLASS=X,CLASS=D,NOTIFY=TSDEAG1
//*****************************************************************
//* ORDENACIÓN FICHEROS DEL MODELO 290
//*****************************************************************
//PASO01   EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SORTJNF1 DD DSN=TSDE.FX.AN.FXJXML03.XMLM290.P1.DIA30,DISP=SHR
//SORTJNF2 DD DSN=TSDE.AG1.PRUEBA.ELIMINAR,DISP=SHR
//SORTJNF3 DD DUMMY
//BOTH     DD DSN=TSDE.CRUCE.CONT.PERCONT.NEW,
//          DISP=(,CATLG,DELETE),
//          SPACE=(20,(90,9),RLSE),STORCLAS=SCDEPER,
//          RECFM=FB,LRECL=004
//F1ONLY   DD DSN=TSDE.CRUCE.CONT.PERCONT.F1,
//          DISP=(,CATLG,DELETE),
//          SPACE=(20,(9,9),RLSE),STORCLAS=SCDEPER,
//          LRECL=223,RECFM=FB
//F2ONLY   DD DSN=TSDE.CRUCE.CONT.PERCONT.F2,
//          DISP=(,CATLG,DELETE),
//          SPACE=(20,(9,9),RLSE),STORCLAS=SCDEPER,
//          LRECL=072,RECFM=FB
//SYSIN    DD *
 JOINKEYS FILE=F1,FIELDS=(463,22,A)
 JOINKEYS FILE=F2,FIELDS=(1,22,A)
 REFORMAT FIELDS=(F1:1,500,?)
 JOIN UNPAIRED,F1,F2
 OPTION COPY
 OUTFIL FNAMES=BOTH,INCLUDE=(296,1,CH,EQ,C'B'),
         BUILD(1,500)
  OUTFIL FNAMES=F1ONLY,INCLUDE=(296,1,CH,EQ,C'1'),
         BUILD(1,500)
  OUTFIL FNAMES=F2ONLY,INCLUDE=(296,1,CH,EQ,C'2'),
         BUILD(1,500)
//*****************************
//*  Programa de validación del modelo 195 facilitado por
//*  la Agencia Tributaria Española
//*************************
//PASO02    EXEC PGM=PRV19509
//STEPLIB   DD  DISP=SHR,DSN=librería de LOAD
//ENTRADA   DD  DSN=TSDE.CRUCE.CONT.PERCONT.F2,DISP=SHR
//SALIDA    DD  DSN=TSDE.CRUCE.CONT.SALIDA,
//          UNIT=3390,DISP=(,CATLG,UNCATLG),
//          DCB=(RECFM=FB,LRECL=418),
//          SPACE=(CYL,(1,1),RLSE)
//IMPRE     DD  DSN=TSDE.AS.IMPRE,
//          UNIT=3390,DISP=(,CATLG,UNCATLG),
//          DCB=(RECFM=FB,LRECL=133),
//          SPACE=(CYL,(1,1),RLSE)
//SYSOUT    DD  SYSOUT=*
//SYSPRINT  DD  SYSOUT=*
```

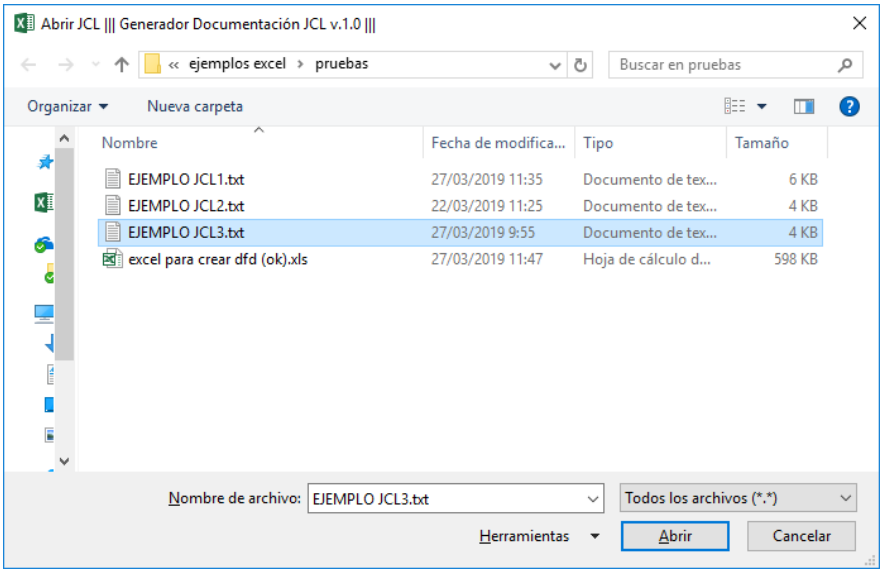
**Datos de salida esperados**

Debería seleccionar los siguientes elementos del proceso:

| Tipo objeto | Nombre | Comentarios |
|---|---|---|
| Paso | PASO01 | Uno de los 2 pasos del JCL |
| Paso | PASO02 | Uno de los 2 pasos del JCL |
| Programa | SORT | Programa ejecutado en el primer paso |
| Programa | PRV19509 | Programa ejecutado en el segundo paso |
| Fichero E | SORTJNF1 | Fichero de entrada del paso01 |
| Fichero E | SORTJNF2 | Fichero de entrada del paso01 |
| Fichero E | SORTJNF3 | Fichero de entrada del paso01 |
| Fichero E | ENTRADA | Fichero de entrada del paso02 |
| Fichero S | BOTH | Fichero de salida paso01 |
| Fichero S | F1ONLY | Fichero de salida paso01 |
| Fichero S | F2ONLY | Fichero de salida paso01 |
| Fichero S | SALIDA | Fichero de salida paso02 |
| Impreso | IMPRE | Impreso de salida paso02 |
| | | |
| | | |

**ANÁLISIS DE LOS RESULTADOS**

Selección del archivo con el código del JCL (EJEMPLO JCL3.txt):




Obtención y visualización del metamodelo con los elementos del proceso:

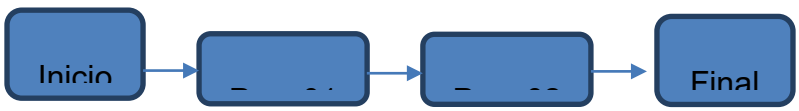
| | A | B | C | D | E | F | G | H | I | J | K | L | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Job | Paso | numPaso | Programa | Fic | DSN | Entrada | Salida | RECFM | LRECL | PARM | DISP | COMENTARIOS |
| 2 | AG1SORT | PASO01 | 1 | SORT | SORTJNF1 | TSDE.FX.AN.FXJXML03.XMLM290.P1.DIA30 | VERDADERO | FALSO | | 0 | | (SHR,KEEP,KEEP) | ORDENACIÓN FICHEROS DE |
| 3 | AG1SORT | PASO01 | 1 | SORT | SORTJNF2 | TSDE.AG1.PRUEBA.ELIMINAR | VERDADERO | FALSO | | 0 | | (SHR,KEEP,KEEP) | |
| 4 | AG1SORT | PASO01 | 1 | SORT | SORTJNF3 | DUMMY6[SORTJNF3] | VERDADERO | FALSO | | 0 | | (SHR,KEEP,KEEP) | |
| 5 | AG1SORT | PASO01 | 1 | SORT | BOTH | TSDE.CRUCE.CONT.PERCONT.NEW | FALSO | VERDADERO | FB | 4 | | (NEW,CATLG,DELETE) | |
| 6 | AG1SORT | PASO01 | 1 | SORT | F1ONLY | TSDE.CRUCE.CONT.PERCONT.F1 | FALSO | VERDADERO | FB | 223 | | (NEW,CATLG,DELETE) | |
| 7 | AG1SORT | PASO01 | 1 | SORT | F2ONLY | TSDE.CRUCE.CONT.PERCONT.F2 | FALSO | VERDADERO | FB | 72 | | (NEW,CATLG,DELETE) | |
| 8 | AG1SORT | PASO01 | 1 | SORT | SYSIN | [1]-PASO01 [SYSIN] | VERDADERO | FALSO | | 0 | | (NEW,CATLG,DELETE) | |
| 9 | AG1SORT | PASO02 | 2 | PRV19509 | ENTRADA | TSDE.CRUCE.CONT.PERCONT.F2 | VERDADERO | FALSO | | 0 | | (SHR,KEEP,KEEP) | Programa de validación de |
| 10 | AG1SORT | PASO02 | 2 | PRV19509 | SALIDA | TSDE.CRUCE.CONT.SALIDA | FALSO | VERDADERO | FB | 418 | | (NEW,CATLG,UNCATLG) | |
| 11 | AG1SORT | PASO02 | 2 | PRV19509 | IMPRE | TSDE.AS.IMPRE | FALSO | VERDADERO | FB | 133 | | (NEW,CATLG,UNCATLG) | |
| 12 | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | |

Se verificar que los elementos obtenidos corresponden con los resultados esperados

**RESULTADO OK**

| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 05/04/2019 | Definición de la prueba 1h |
| | 05/04/2019 | Preparación de la prueba 2h |
| | 05/04/2019 | Finalizada la prueba |

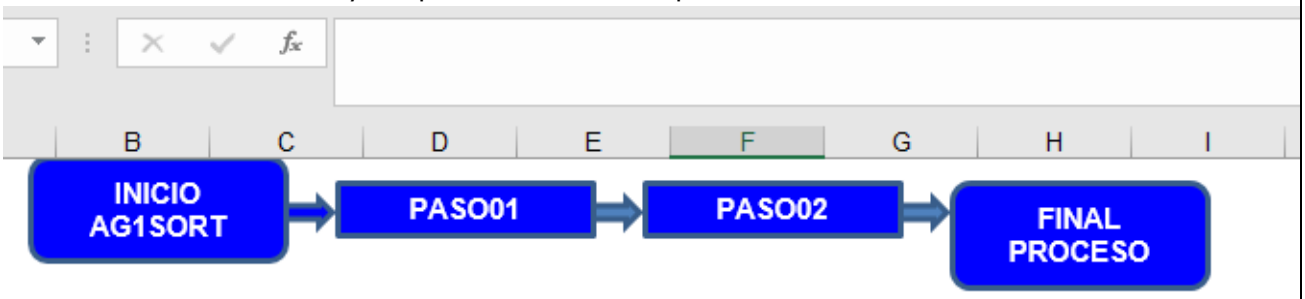
*Tabla 79. Detalle caso de prueba CASO-U-001*

| **ID:** | CASO-U-002 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad/ Requisitos** | **Validación del sub requisito REQ-001-02** Creación de un diagrama de flujo de datos de ejecución (nivel1) | | |
| **Descripción** | Comprobar el correcto diseño del diagrama de ejecución, el DFD nivel 1 partiendo del código del proceso JCL | | |
| **Datos de entrada** | Se utiliza el mismo código JCL de la prueba CASO-U-001 que contenía 2 pasos. | | |
| **Datos de salida esperados** | El resultado gráfico resultante debería ser parecido al siguiente:   | | |

**ANÁLISIS DE LOS RESULTADOS**

Selección del archivo con el código del JCL (EJEMPLO JCL3.txt).

Visualización del DFD nivel 1 y comparación con datos esperados:

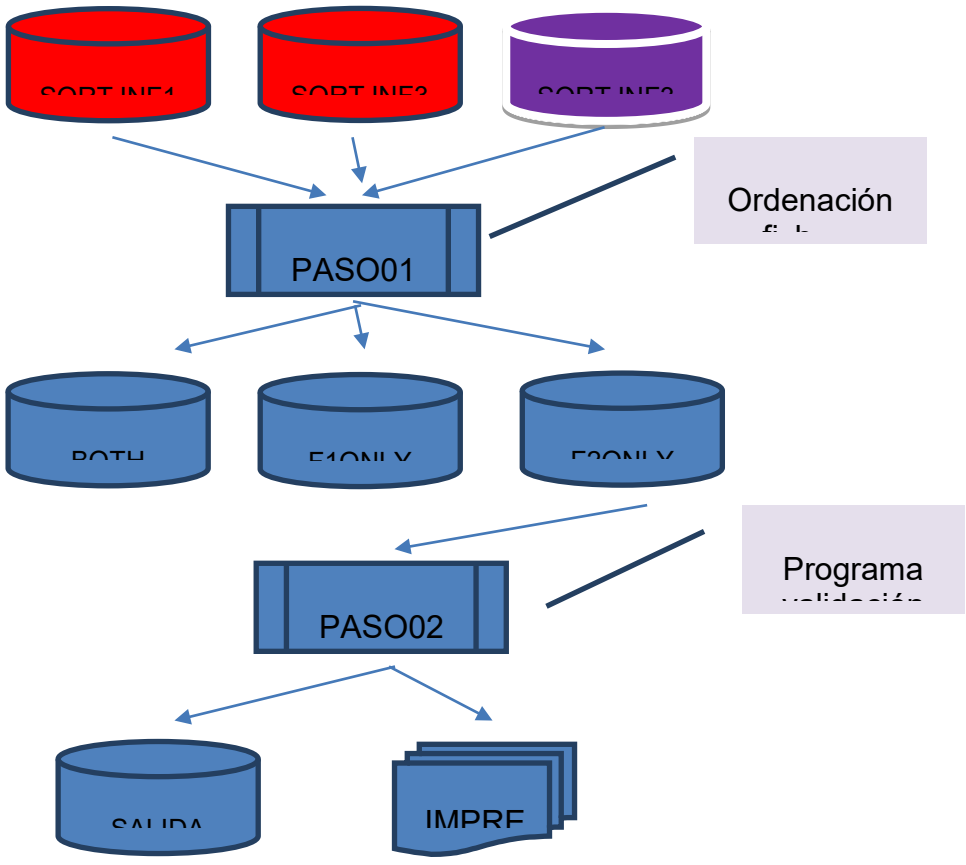



Resultado esperado correcto.

**RESULTADO OK**

| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 05/04/2019 | Definición de la prueba 1h |
| | 05/04/2019 | Preparación de la prueba 2h |
| | 05/04/2019 | Finalizada la prueba |

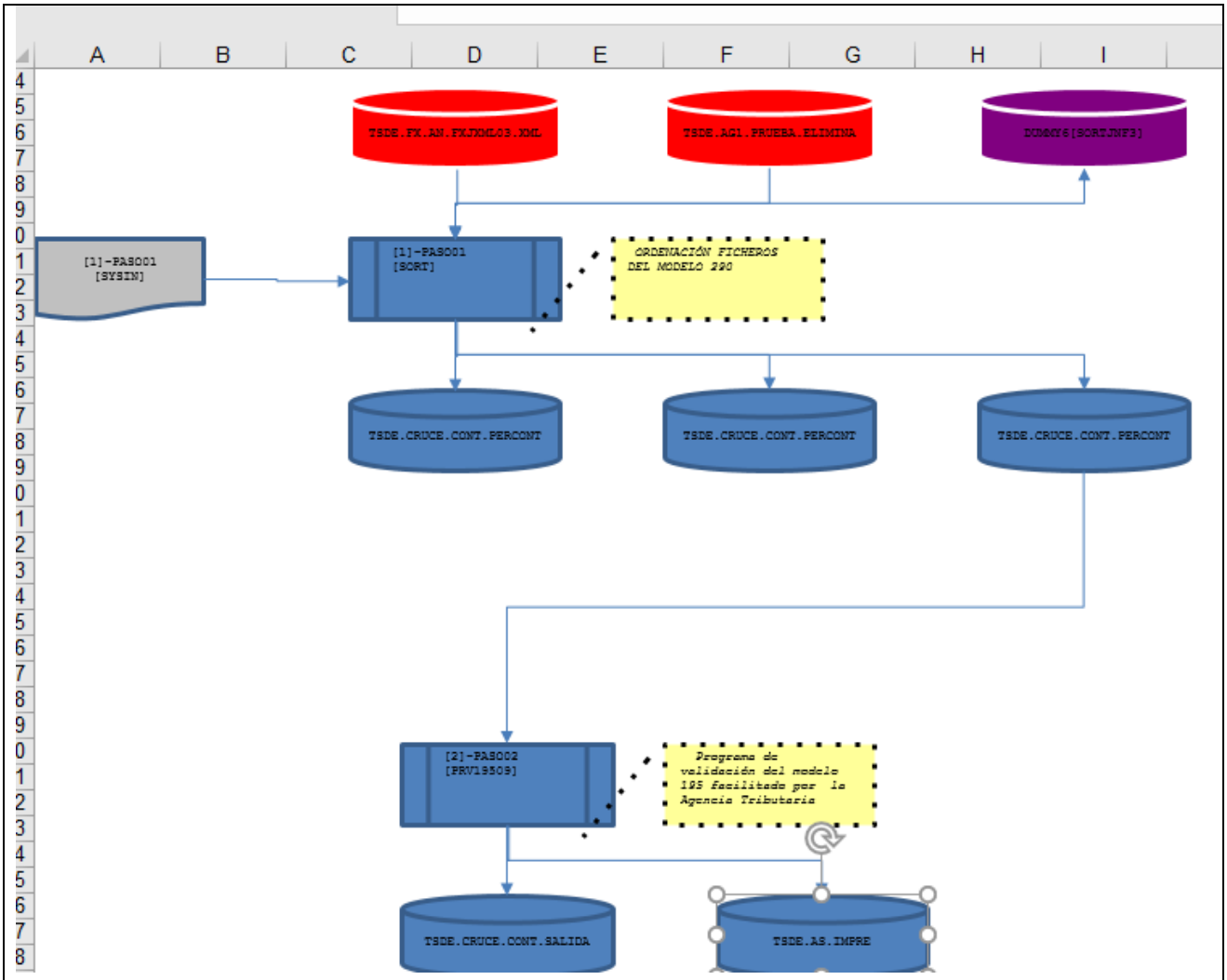
*Tabla 80. Detalle caso de prueba CASO-U-002*

| | | | |
|---|---|---|---|
| **ID:** | CASO-U-003 | **Responsable:** | Abel |
| **Funcionalidad / Requisitos** | **Validación del sub requisito REQ-001-03** Creación de un diagrama de flujo de datos de ejecución (nivel2) | | |
| **Descripción** | Comprobar el correcto diseño del diagrama de ejecución, el DFD nivel 2 partiendo del código del proceso JCL | | |
| **Datos de entrada** | Se utiliza el mismo código JCL de la prueba CASO-U-001 que contenía 2 pasos. | | |
| **Datos de salida esperados** | El resultado gráfico resultante debería ser parecido al siguiente:<br>  | | |

**ANÁLISIS DE LOS RESULTADOS**

Selección del archivo con el código del JCL (EJEMPLO JCL3.txt):

Visualización del DFD nivel 2 y comparación con datos esperados:




**RESULTADO OK**

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 06/04/2019 | Definición de la prueba 2h |
| | 06/04/2019 | Preparación de la prueba 3h |
| | 06/04/2019 | Finalizada la prueba |

*Tabla 81. Detalle caso de prueba CASO-U-003*

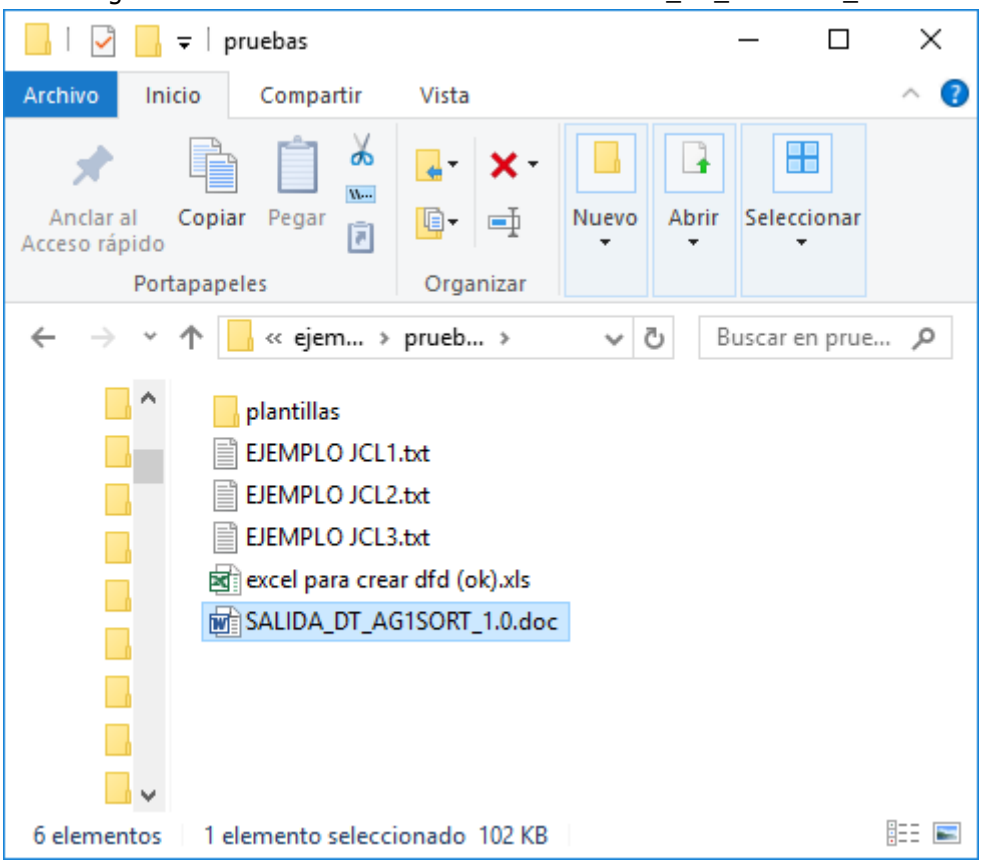
| ID: | CASO-U-004 | Responsable: | Abel |
|---|---|---|---|
| Funcionalidad / Requisitos | **Validación del sub requisito REQ-001-04** Creación de un diagrama de flujo de datos de ejecución (nivel1) | | |
| Descripción | Creación del documento DT según plantilla y nombre especificado. | | |
| Datos de entrada | Se utiliza el mismo código JCL de la prueba CASO-U-001 que contenía 2 pasos. | | |

| | |
|---|---|
| **Datos de salida esperados** | Se debe generar un documento Word llamado SALIDA_DT_AG1SORT_1.0.doc que contenga el formato especificado en la *'Tabla nn. Formato documento DT proceso JCL'*<br>Recordemos los apartados principales:<br>**Índice**<br>**HOJA DE CONTROL**<br>**1 PROPÓSITO DEL DOCUMENTO**<br>**2 ARQUITECTURA DE COMPONENTES (HW Y SW)**<br>**3 DIAGRAMA DE EJECUCIÓN (DFD NIVEL 1)**<br>**4 DIAGRAMA DETALLADO (DFD NIVEL 2)**<br>**5 DESCRIPCIÓN DETALLADA DE COMPONENTES**<br>**6 DOCUMENTOS RELACIONADOS**<br>**ANEXOS** |

**ANÁLISIS DE LOS RESULTADOS**

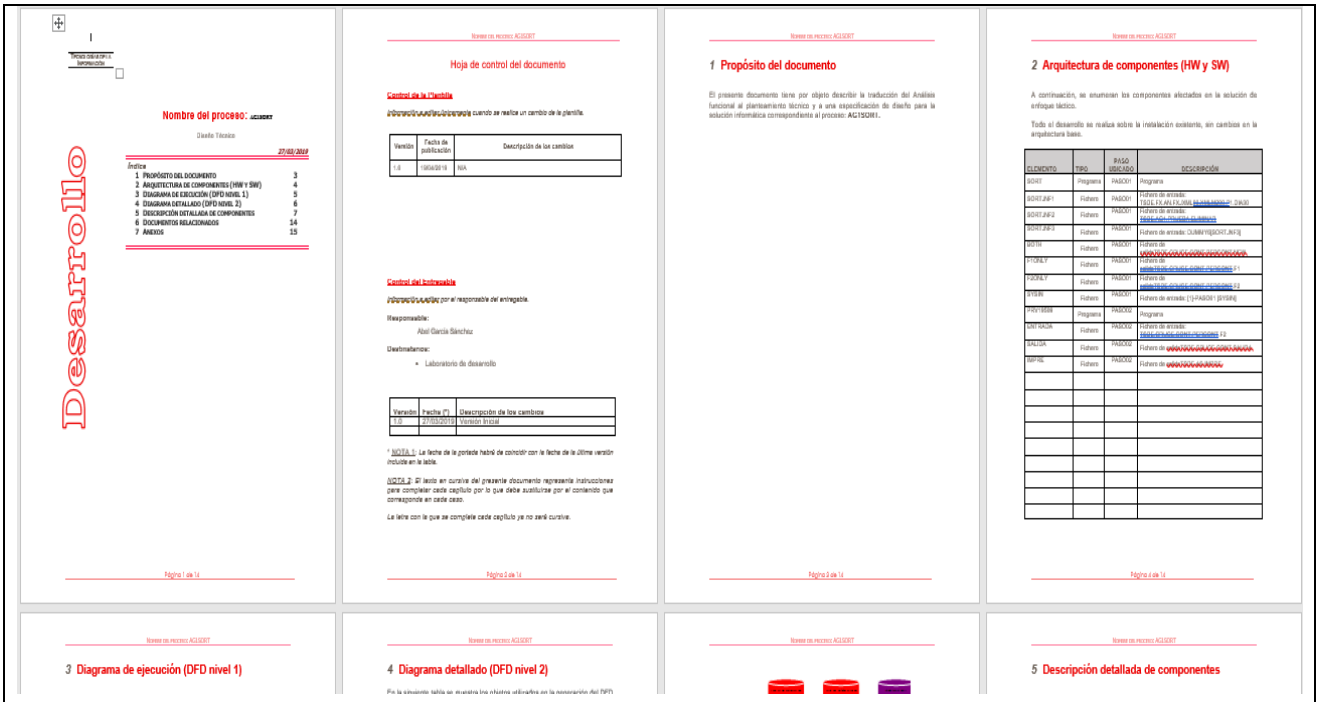
Selección del archivo con el código del JCL (EJEMPLO JCL3.txt):

Verificar generación del documento Word con el SALIDA_DT_AG1SORT_1.0.doc:




OK

Verificar la generación de los apartados definidos y la estructura del documento:




**RESULTADO OK**

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 06/04/2019 | Definición de la prueba 1h |
| | 06/04/2019 | Preparación de la prueba 2h |
| | 06/04/2019 | Finalizada la prueba |

*Tabla 82. Detalle caso de prueba CASO-U-004*

| ID: | CASO-I-001 | Responsable: | Abel |
|---|---|---|---|
| Funcionalidad / Requisitos | **Validación del requisito REQ-001** Generación automática de documentación técnica sobre procesos JCL | | |
| Descripción | Prueba íntegra de todas las funcionalidades definidas en el requisito REQ-001. | | |
| Datos de entrada | Se utiliza el mismo código JCL de la prueba CASO-U-001 que contenía 2 pasos. | | |
| Datos de salida esperados | Generación de la siguiente información:<br>• Detalle de elementos del proceso JCL (definidos en CASO-U-001)<br>• Diseño DFD nivel 1 (definidos en CASO-U-002)<br>• Diseño DFD nivel 2 (definidos en CASO-U-003)<br>• Generación del documento DT con las siguientes características: | | |

- Formato del documento según especificaciones.
- Relación de la arquitectura de componentes.
- Inclusión del gráfico del DFD nivel 1.
- Inclusión del gráfico del DFD nivel 2.
- Detalle de los pasos del JCL

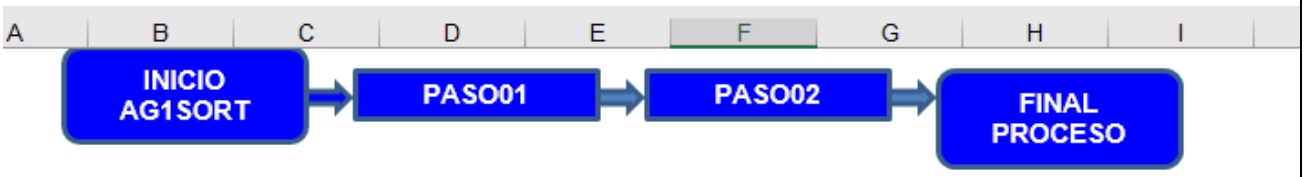
**ANÁLISIS DE LOS RESULTADOS**

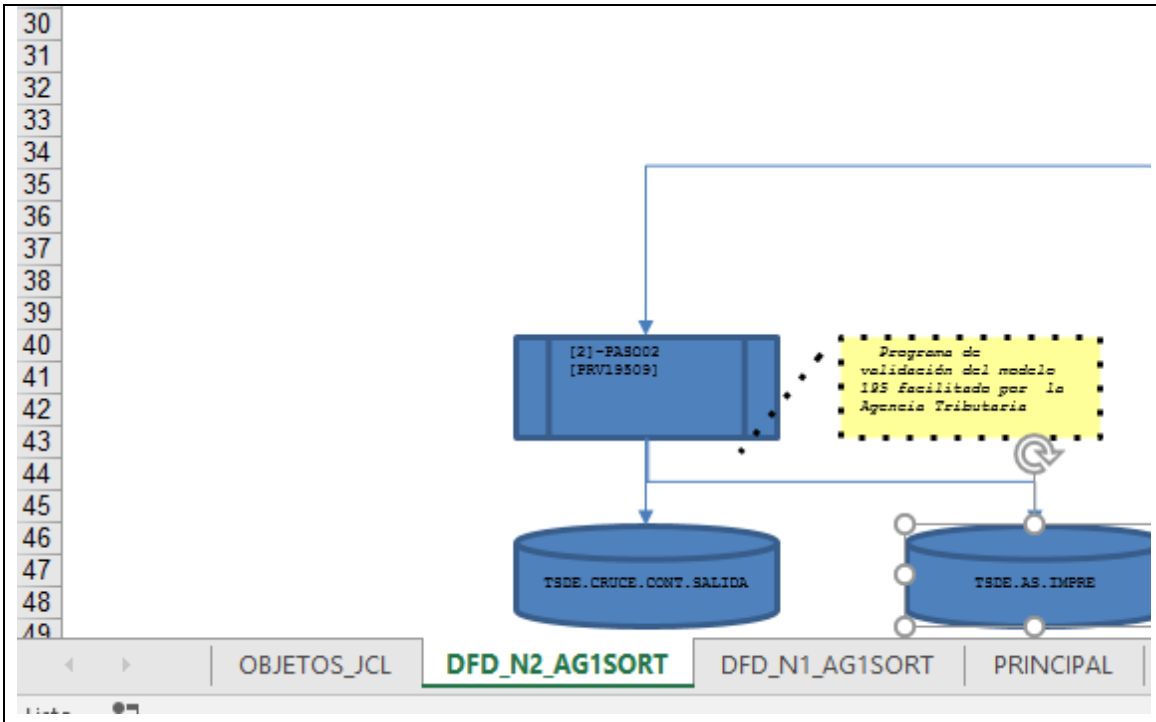
Selección del archivo con el código del JCL (EJEMPLO JCL3.txt):

Comprobar componentes generados de forma individual:

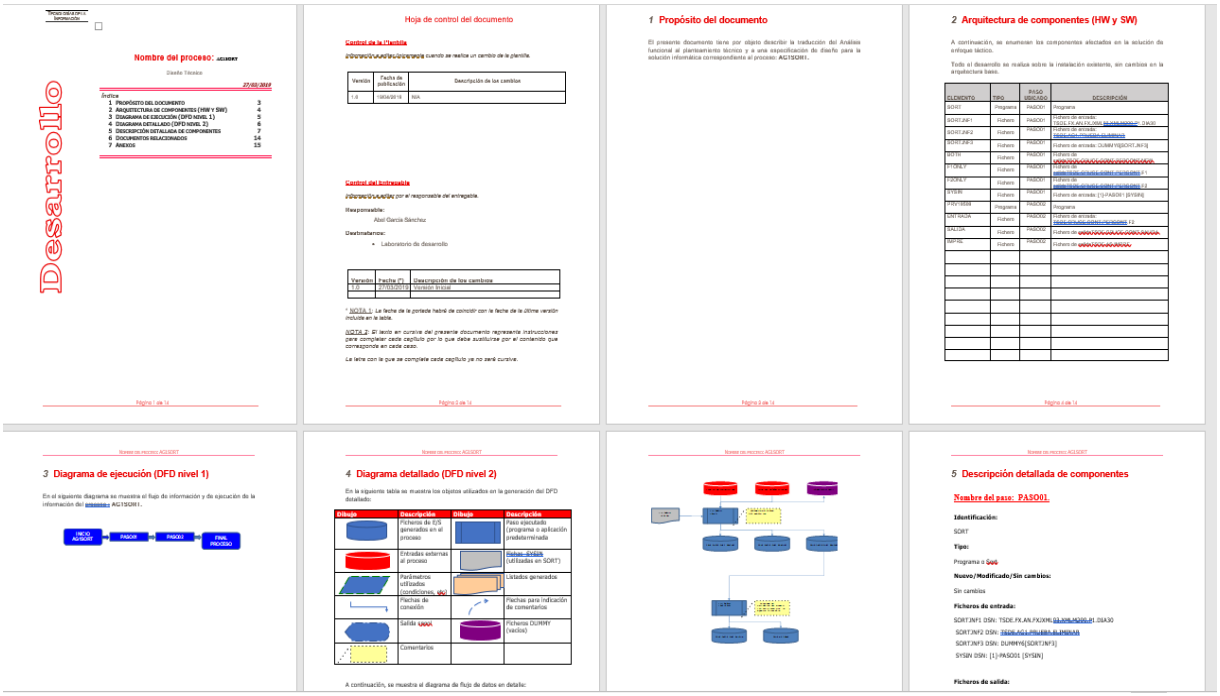
| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Job | Paso | numPaso | Programa | Fic | DSN |
| 2 | AG1SORT | PASO01 | 1 | SORT | SORTJNF1 | TSDE.FX.AN.F> |
| 3 | AG1SORT | PASO01 | 1 | SORT | SORTJNF2 | TSDE.AG1.PRU |
| 4 | AG1SORT | PASO01 | 1 | SORT | SORTJNF3 | DUMMY6[SORT |
| 5 | AG1SORT | PASO01 | 1 | SORT | BOTH | TSDE.CRUCE.C |
| 6 | AG1SORT | PASO01 | 1 | SORT | F1ONLY | TSDE.CRUCE.C |
| 7 | AG1SORT | PASO01 | 1 | SORT | F2ONLY | TSDE.CRUCE.C |
| 8 | AG1SORT | PASO01 | 1 | SORT | SYSIN | [1]-PASO01 [SYSIN] |
| 9 | AG1SORT | PASO02 | 2 | PRV19509 | ENTRADA | TSDE.CRUCE.C |
| 10 | AG1SORT | PASO02 | 2 | PRV19509 | SALIDA | TSDE.CRUCE.C |
| 11 | AG1SORT | PASO02 | 2 | PRV19509 | IMPRE | TSDE.AS.IMPR |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |
| 17 | | | | | | |
| 18 | | | | | | |
| 19 | | | | | | |
| 20 | | | | | | |
| 21 | | | | | | |
| 22 | | | | | | |
| 23 | | | | | | |
| 24 | | | | | | |
| 25 | | | | | | |
| 26 | | | | | | |
| 27 | | | | | | |
| 28 | | | | | | |
| 29 | | | | | | |
| 30 | | | | | | |
| 31 | | | | | | |
| 32 | | | | | | |

OBJETOS_JCL | DFD_N2_AG1SORT | DFD_N1_AG1SORT






Y documento con todos los elementos incrustados y generados:




Documento:

SALIDA_DT_AG1SORT_1.0.doc

Resultado OK

Verificar la generación del DT con todas las características de éste:

**RESULTADO OK**

| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 06/04/2019 | Definición de la prueba 2h |
| | 06/04/2019 | Preparación de la prueba 3h |
| | 06/04/2019 | Finalizada la prueba |

*Tabla 83. Detalle caso de prueba CASO-I-001*

| **ID:** | CASO-I-002 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad / Requisitos** | **Validación del requisito REQ-001** Generación automática de documentación técnica sobre procesos JCL | | |
| **Descripción** | Prueba íntegra de todas las funcionalidades definidas en el requisito REQ-001. | | |
| **Datos de entrada** | Se utiliza un JCL con más pasos, en concreto 4 pasos. El archivo es EJEMPLO JCL1.txt | | |
| **Datos de salida esperados** | Los datos esperados son los mismos que en el CASO-I-001, pero con más elementos software, más pasos, DFD diferentes, etc. | | |

**ANÁLISIS DE LOS RESULTADOS**

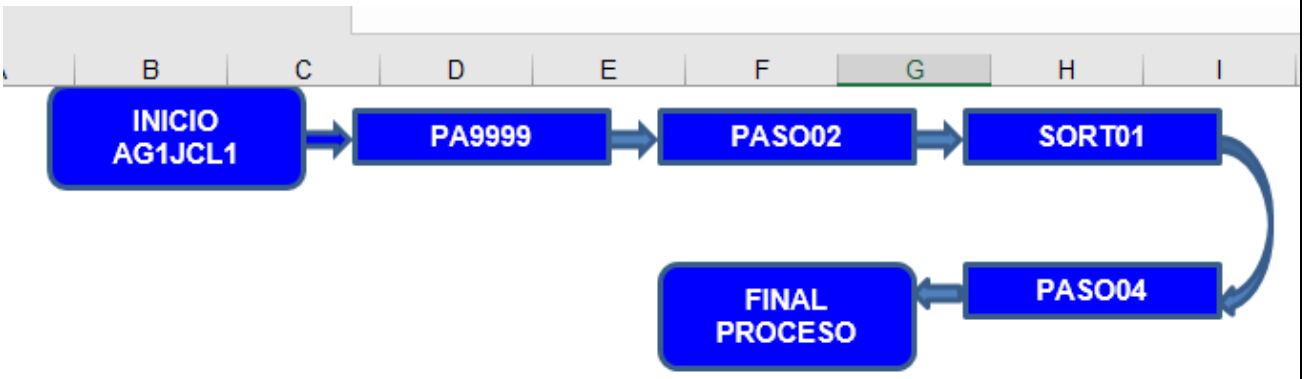
Selección del archivo con el código del JCL (EJEMPLO JCL1.txt):

Comprobar componentes generados de forma individual:
Elementos OK

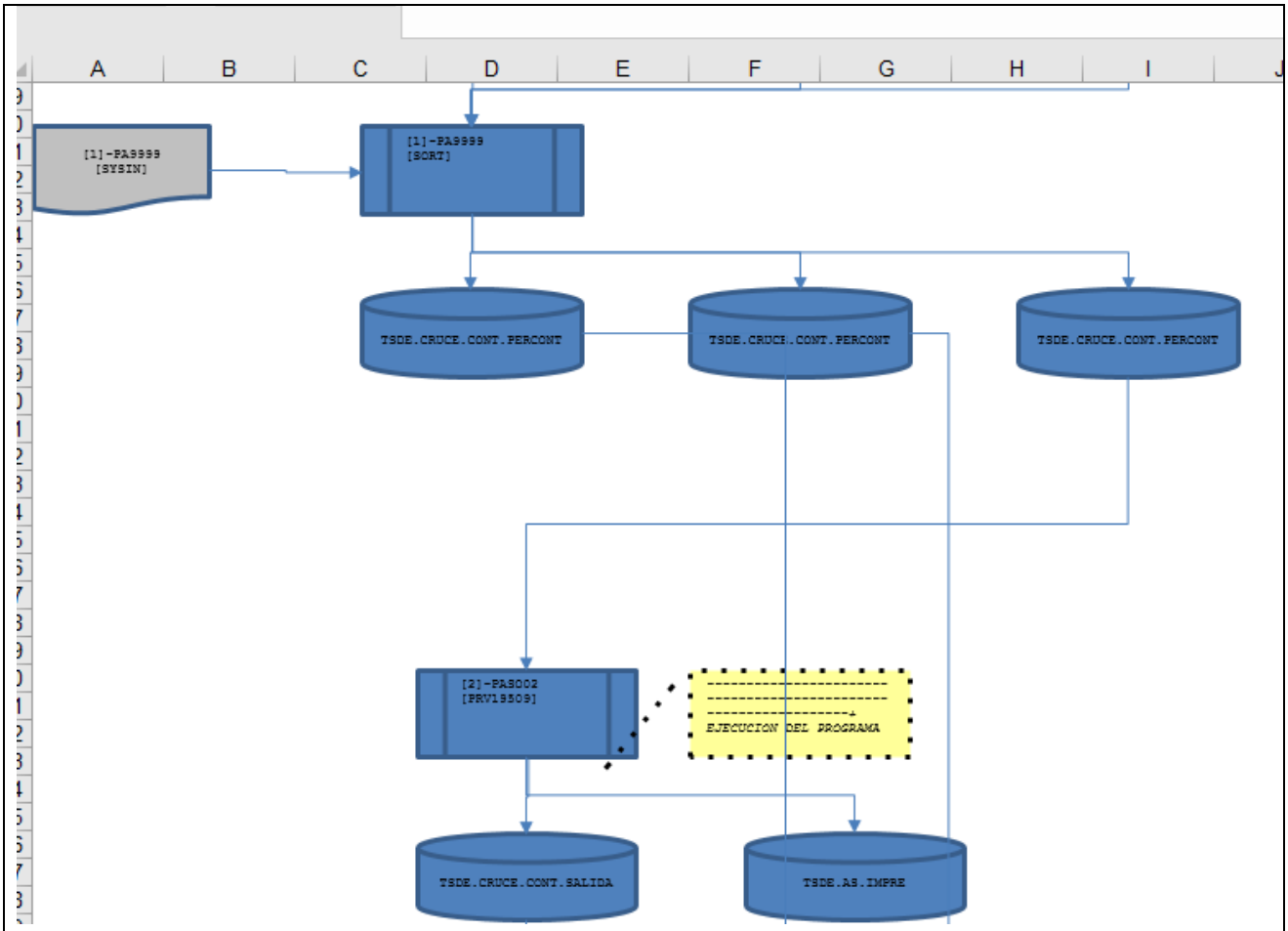
| Job | Paso | numPaso | Programa | ETC |
|---|---|---|---|---|
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PA9999 | 1 | SORT | ... |
| AG1JCL1 | PASO02 | 2 | PRV19509 | ... |
| AG1JCL1 | PASO02 | 2 | PRV19509 | ... |
| AG1JCL1 | PASO02 | 2 | PRV19509 | ... |
| AG1JCL1 | SORT01 | 3 | SORT | ... |
| AG1JCL1 | SORT01 | 3 | SORT | ... |

| | | | | |
|---|---|---|---|---|
| AG1JCL1 | SORT01 | 3 | SORT | ... |
| AG1JCL1 | SORT01 | 3 | SORT | ... |
| AG1JCL1 | SORT01 | 3 | SORT | ... |
| AG1JCL1 | PASO04 | 4 | PRV19505 | ... |
| AG1JCL1 | PASO04 | 4 | PRV19505 | ... |
| AG1JCL1 | PASO04 | 4 | PRV19505 | ... |

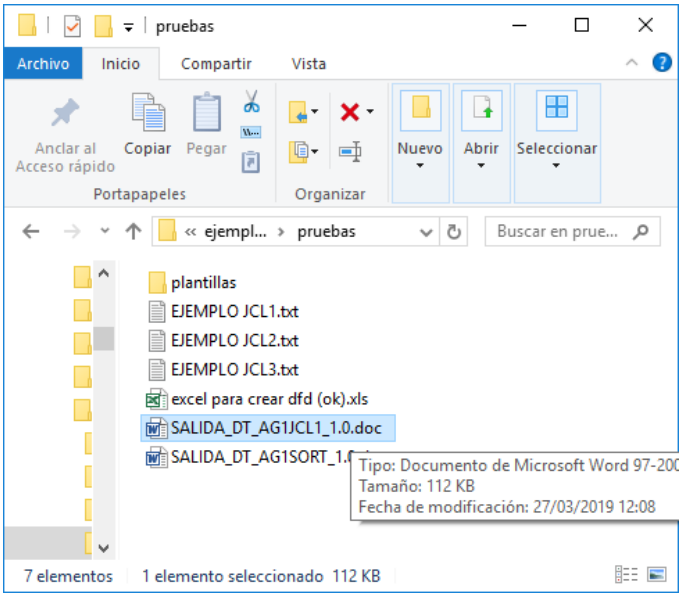
DFD NIVEL 1 OK




DFD detalle OK




Documento DT ok



## 2 Arquitectura de componentes (HW y SW)

A continuación, se enumeran los componentes afectados en la solución de enfoque táctico.

Todo el desarrollo se realiza sobre la instalación existente, sin cambios en la arquitectura base.

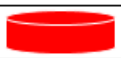
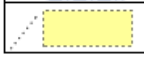
| ELEMENTO | TIPO | PASO UBICADO | DESCRIPCIÓN |
|---|---|---|---|
| SORT | Programa | PA9999 | Programa |
| SORTJNF1 | Fichero | PA9999 | Fichero de entrada: TSDE.FX.AN.FXJXML03.XMLM280.R1.DIA30 |
| SORTJNF2 | Fichero | PA9999 | Fichero de entrada: TSDE.AG1.PRUEBA.ELIMINAR |
| SORTJNF3 | Fichero | PA9999 | Fichero de entrada: DUMMY8[SORTJNF3] |
| BOTH | Fichero | PA9999 | Fichero de salidaTSDE.CRUCE.CONT.PERCONT.NEW |
| F1ONLY | Fichero | PA9999 | Fichero de salidaTSDE.CRUCE.CONT.PERCONT.F1 |
| F2ONLY | Fichero | PA9999 | Fichero de salidaTSDE.CRUCE.CONT.PERCONT.F2 |
| SYSIN | Fichero | PA9999 | Fichero de entrada: [1]-PA9999 [SYSIN] |
| PRV19509 | Programa | PASC02 | Programa |
| ENTRADA | Fichero | PASC02 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.F2 |
| SALIDA | Fichero | PASC02 | Fichero de salidaTSDE.CRUCE.CONT.SALIDA |
| IMPRE | Fichero | PASC02 | Fichero de salidaTSDE.AS.IMPRE |
| SORT | Programa | SORT01 | Programa |
| SORTIN | Fichero | SORT01 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.F1 |
| SORTINZ | Fichero | SORT01 | Fichero de entrada: TSDE.CRUCE.CONT.SALIDA |
| SORTOUT | Fichero | SORT01 | Fichero de salidaTSDE.FX.AN.FXJ18901.MODEL189.ALLIANZ.PARRU |
| SYSIN | Fichero | SORT01 | Fichero de entrada: [3]-SORT01 [SYSIN] |
| SYSIN | Fichero | SORT01 | Fichero de entrada: [3]-SORT01 [SYSIN] |
| PRV19505 | Programa | PASC04 | Programa |
| CINTA | Fichero | PASC04 | Fichero de entrada: TSDE.CRUCE.CONT.PERCONT.NEW |
| SALIDA | Fichero | PASC04 | Fichero de salidaTSDE.SALIDA.ERRORES |
| IMPRE | | | Fichero de salidaTSDE.FICHIER.PARA.IMERI |

## 3 Diagrama de ejecución (DFD nivel 1)

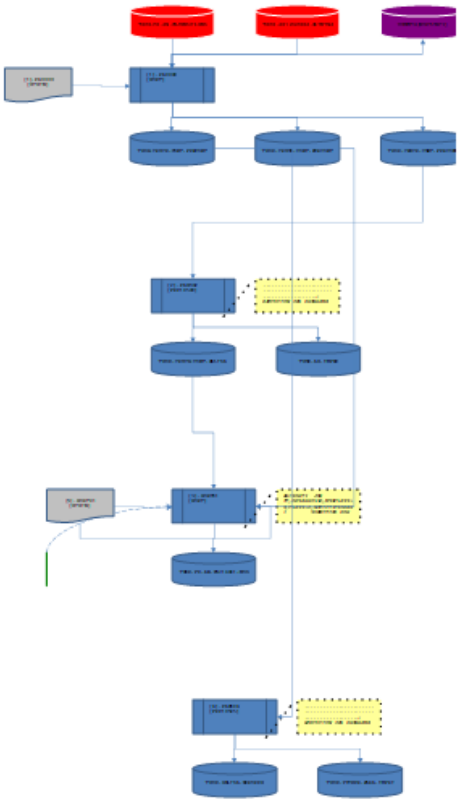
En el siguiente diagrama se muestra el flujo de información y de ejecución de la información del proceso : AG1JCL1.

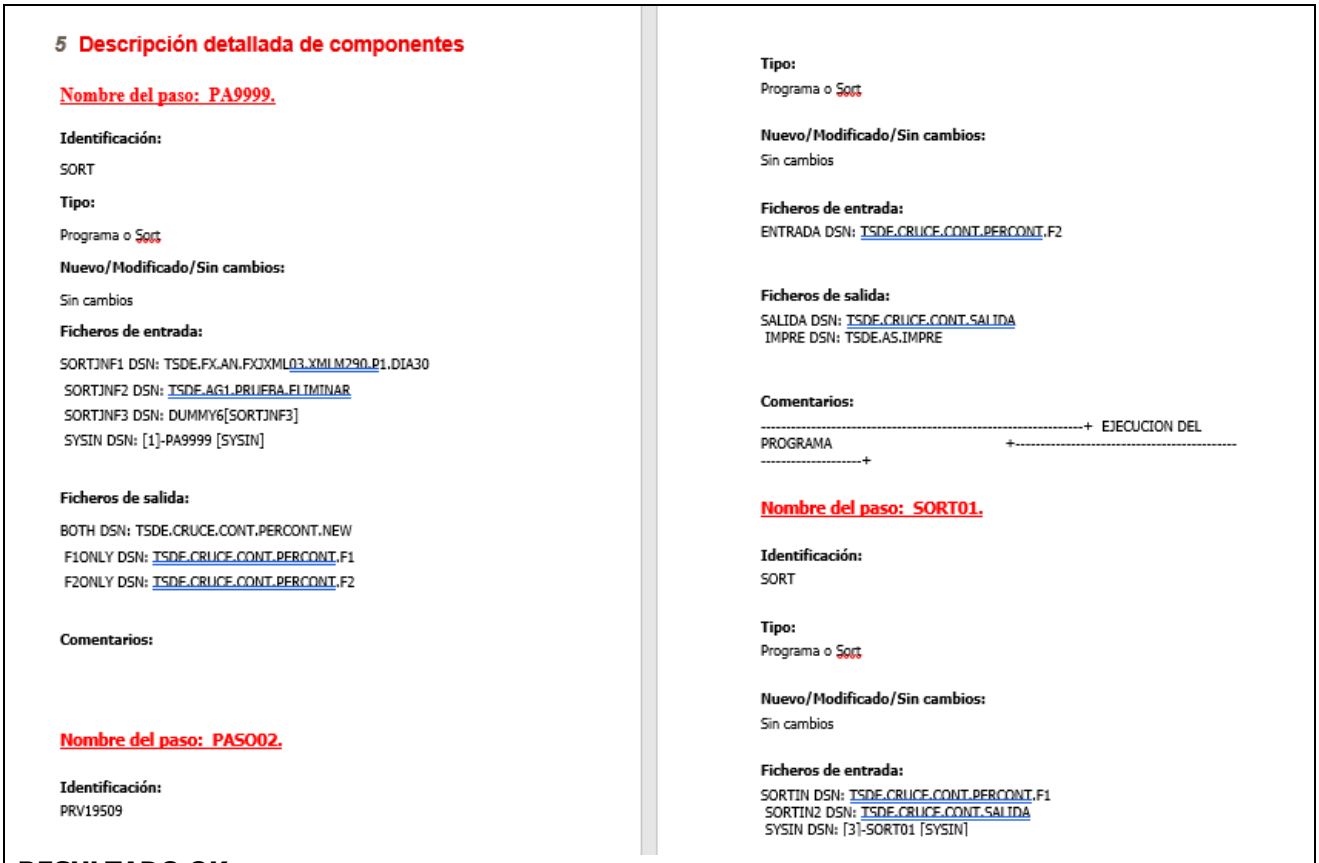

## 4 Diagrama detallado (DFD nivel 2)

En la siguiente tabla se muestra los objetos utilizados en la generación del DFD detallado:

| Dibujo | Descripción | Dibujo | Descripción |
|---|---|---|---|
|  | Ficheros de E/S generados en el proceso |  | Paso ejecutado (programa o aplicación predeterminada |
|  | Entradas externas al proceso |  | Fichas SYSIN (utilizadas en SORT) |
|  | Parámetros utilizados (condiciones, etc) |  | Listados generados |
|  | Flechas de conexión |  | Flechas para indicación de comentarios |
|  | Salida spool |  | Ficheros DUMMY (vacíos) |
|  | Comentarios | | |

A continuación, se muestra el diagrama de flujo de datos en detalle:





5 Descripción detallada de componentes

Nombre del paso: PA9999.

**Identificación:**
SORT

**Tipo:**
Programa o Sort

**Nuevo/Modificado/Sin cambios:**
Sin cambios

**Ficheros de entrada:**
SORTJNF1 DSN: TSDE.FX.AN.FXJXML03.XMLM290.P1.DIA30
SORTJNF2 DSN: TSDE.AG1.PRUEBA.ELIMINAR
SORTJNF3 DSN: DUMMY6[SORTJNF3]
SYSIN DSN: [1]-PA9999 [SYSIN]

**Ficheros de salida:**
BOTH DSN: TSDE.CRUCE.CONT.PERCONT.NEW
F1ONLY DSN: TSDE.CRUCE.CONT.PERCONT.F1
F2ONLY DSN: TSDE.CRUCE.CONT.PERCONT.F2

**Comentarios:**

Nombre del paso: PASO02.

**Identificación:**
PRV19509

**Tipo:**
Programa o Sort

**Nuevo/Modificado/Sin cambios:**
Sin cambios

**Ficheros de entrada:**
ENTRADA DSN: TSDE.CRUCE.CONT.PERCONT.F2

**Ficheros de salida:**
SALIDA DSN: TSDE.CRUCE.CONT.SALIDA
IMPRE DSN: TSDE.AS.IMPRE

**Comentarios:**
----------+ EJECUCION DEL
PROGRAMA +----------
----------+

Nombre del paso: SORT01.

**Identificación:**
SORT

**Tipo:**
Programa o Sort

**Nuevo/Modificado/Sin cambios:**
Sin cambios

**Ficheros de entrada:**
SORTIN DSN: TSDE.CRUCE.CONT.PERCONT.F1
SORTIN2 DSN: TSDE.CRUCE.CONT.SALIDA
SYSIN DSN: [3]-SORT01 [SYSIN]

**RESULTADO OK**

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 06/04/2019 | Definición de la prueba 2h |
| | 06/04/2019 | Preparación de la prueba 3h |
| | 06/04/2019 | Finalizada la prueba |

*Tabla 84. Detalle caso de prueba CASO-I-002*

| ID: | CASO-O-004 | Responsable: | Abel |
|---|---|---|---|
| Funcionalidad/ Requisitos | **Funcionalidad: Parse código COBOL** | | |
| Descripción | Verificación de la correcta extracción del código COBOL y transformación en los metamodelos definidos. | | |
| Datos de entrada | Código para analizar:<br>`IDENTIFICATION DIVISION.`<br>`PROGRAM-ID. FXP19318.`<br>`ENVIRONMENT DIVISION.`<br>`CONFIGURATION SECTION.`<br>`AUTHOR. AbelGarcia Sanchez.`<br>`SPECIAL-NAMES.`<br>`DECIMAL-POINT IS COMMA.`<br>`WORKING-STORAGE SECTION.`<br>`01 VARIABLE1 PIC X(15).`<br>`01 VARIABLE2 PIC XX.`<br>`01 VARIABLE3 PIC 9(15).`<br>`01 VARIABLE4 PIC 9999.` | | |

```
77 I            PIC 9(5).
PROCEDURE DIVISION.
INICIO.
    MOVE 5 TO VARIABLE3
    MOVE 15 TO VARIABLE4
    COMPUTE RESULTADO = VARIABLE3 * VARIABLE4
    DISPLAY "EL RESULTADO ES: "
    DISPLAY RESULTADO
    STOP RUN.
```

**Datos de salida esperados**

Debería seleccionar los siguientes elementos del proceso:
Para la estructura de objetos, debería obtener 6 objetos → 
FXP19318 (nombre programa)
VARIABLE1 (variable)
VARIABLE2 (variable)
VARIABLE3 (variable)
VARIABLE4 (variable)
I (variable)

Y para la estructura de líneas de código 21 registros estructurados con su información correspondientemente formateada (sentencia, variables utilizadas, secuencia, etc)

**ANÁLISIS DE LOS RESULTADOS**

Selección del archivo pr3Operaciones.cob
Ejecución del proceso y revisión de resultados:

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Tipo | Nombre | Definición | Longitud | Observaciones | Linkage | Nivel | | | | | | | | |
| 2 | Nombre del programa | FXP19318 | | 0 | Warning. No utilizado | FALSO | | | | | | | | | |
| 3 | VARIABLE | VARIABLE1 | alfanumerico | 15 | Warning. No utilizado | FALSO | 1 | | | | | | | | |
| 4 | VARIABLE | VARIABLE2 | alfanumerico | 2 | Warning. No utilizado | FALSO | 1 | | | | | | | | |
| 5 | VARIABLE | VARIABLE3 | numerico | 15 | OK. Variable utilizada | FALSO | 1 | | | | | | | | |
| 6 | VARIABLE | VARIABLE4 | numerico | 4 | OK. Variable utilizada | FALSO | 1 | | | | | | | | |
| 7 | VARIABLE | I | numerico | 5 | Warning. No utilizado | FALSO | 77 | | | | | | | | |
| 8 | | | | | | | | | | | | | | | |
| 9 | Entrada | Sentencia | Loc proceso | Localización | Variable 1 | Variable 2 | Variable 3 | Variable 4 | Camino | Camino Máximo | Camino asignado | | | | |
| 10 | 1 | IDENTIFICATION | | IDENTIFICATION | | | | | 1 | 1 | 1 | | | | |
| 11 | 2 | PROGRAM-ID | | IDENTIFICATION | FXP19318 | | | | 1 | 1 | 1 | | | | |
| 12 | 3 | ENVIRONMENT | | IDENTIFICATION | | | | | 1 | 1 | 1 | | | | |
| 13 | 4 | CONFIGURATION | | IDENTIFICATION | | | | | 1 | 1 | 1 | | | | |
| 14 | 5 | AUTHOR | | IDENTIFICATION | AbelGarcia | | | | 1 | 1 | 1 | | | | |
| 15 | 6 | SPECIAL-NAMES | | IDENTIFICATION | | | | | 1 | 1 | 1 | | | | |
| 16 | 7 | DECIMAL-POINT | | IDENTIFICATION | | | | | 1 | 1 | 1 | | | | |
| 17 | 8 | WORKING-STORAGE | | WORKING-STORAGE | | | | | 1 | 1 | 1 | | | | |
| 18 | 9 | 1 | | WORKING-STORAGE | | | | | 1 | 1 | 1 | | | | |
| 19 | 10 | 1 | | WORKING-STORAGE | | | | | 1 | 1 | 1 | | | | |
| 20 | 11 | 1 | | WORKING-STORAGE | | | | | 1 | 1 | 1 | | | | |
| 21 | 12 | 1 | | WORKING-STORAGE | | | | | 1 | 1 | 1 | | | | |
| 22 | 13 | 77 | | WORKING-STORAGE | | | | | 1 | 1 | 1 | | | | |
| 23 | 14 | PROCEDURE | | PROCEDURE | | | | | 1 | 1 | 1 | | | | |
| 24 | 15 | INICIO | | PROCEDURE | | | | | 1 | 1 | 1 | | | | |
| 25 | 16 | MOVE | | PROCEDURE | 5 | TO | VARIABLE3 | | 1 | 1 | 1 | | | | |
| 26 | 17 | MOVE | | PROCEDURE | 15 | TO | VARIABLE4 | | 1 | 1 | 1 | | | | |
| 27 | 18 | COMPUTE | | PROCEDURE | | | | | 1 | 1 | 1 | | | | |
| 28 | 19 | DISPLAY | | PROCEDURE | "EL RESULTADO ES: " | | | | 1 | 1 | 1 | | | | |
| 29 | 20 | DISPLAY | | PROCEDURE | RESULTADO | | | | 1 | 1 | 1 | | | | |

ESTRUCTURA DE OBJETOS

ESTRUCTURA CÓDIGO

El resultado corresponde con la información esperada.
**RESULTADO OK**

| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 07/05/2019 | Definición de la prueba 1h |
| | 07/05/2019 | Preparación de la prueba 2h |
| | 07/05/2019 | Finalizada la prueba |

*Tabla 85. Detalle caso de prueba CASO-O-004*

| **ID:** | CASO-U-006 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad/ Requisitos** | **Validación sub requisito REQ-002-01.** Generar programa llamador en COBOL. | | |
| **Descripción** | Validar la generación correcta de un programa principal en COBOL a partir de los datos de intercambio | | |
| **Datos de entrada** | Análisis de un módulo COBOL con el siguiente área de intercambio:<br>LINKAGE SECTION.<br>01 AREA-INTERCAMBIO.<br>05 OPCION PIC X(10).<br>05 ENTIDAD PIC 9(4).<br>05 IBAN PIC X(24).<br>05 RESPUESTA PIC X(100). | | |
| **Datos de salida esperados** | Se espera que se genera un programa principal COBOL que defina correctamente los datos de trabajo e intercambio, realice una entrada por parámetros de estos datos para poder realizar pruebas, los verifique y si todo es correcto llamé al módulo analizado y visualice la respuesta por consola. | | |

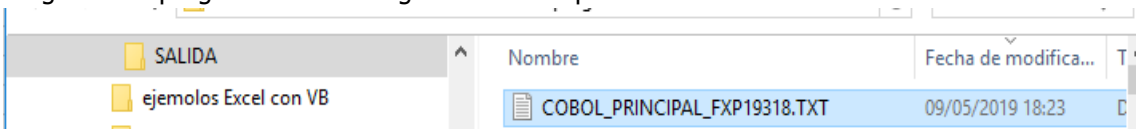
**ANÁLISIS DE LOS RESULTADOS**

Selección del programa pr2 es modulo.cob

pr2 es modulo.cob

Ejecución del proceso:

Se genera el programa COBOL según nombre esperado:




Analizamos el código:

COBOL_PRINCIPAL_ FXP19318.TXT

Definición de las áreas de dato OK

COBOL_PRINCIPAL_FXP19318.TXT: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
       01 TABLA-ERRORES.
          10 W-ERROR OCCURS 100 PIC X(100).
       01 NUMERO-ERRORES        PIC 9(05).
       01 I                     PIC 9(05).
       01 MODULO                PIC X(20).
       01 AUX-AREA-INTERCAMBIO.
           05 AUX-OPCION    PIC X(10).
           05 AUX-ENTIDAD    PIC 9(4).
           05 AUX-IBAN    PIC X(24).
           05 AUX-RESPUESTA    PIC X(100).

      *****************************************************************
      *      PROCESO PRINCIPAL                                        *
      *****************************************************************
       LINKAGE SECTION.
       01 RESERVA-PARM           PIC X(2).
       01 LNK-AREA-INTERCAMBIO.
           05 LNK-OPCION    PIC X(10).
           05 LNK-ENTIDAD    PIC 9(4).
           05 LNK-IBAN    PIC X(24).
           05 LNK-RESPUESTA    PIC X(100).
       PROCEDURE DIVISION USING RESERVA-PARM  LNK-AREA-INTERCAMBIO.
       COMIENZO.
```

Validación de entrada OK

COBOL_PRINCIPAL_FXP19318.TXT: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
        VALIDAR-ENTRADA.
      *--------------------*
            IF AUX-OPCION = SPACES
               ADD 1 TO NUMERO-ERRORES
               MOVE 'DATO NO INFORMADO' TO W-ERROR(NUMERO-ERRORES)
            END-IF.
            IF AUX-ENTIDAD NOT NUMERIC
               ADD 1 TO NUMERO-ERRORES
               MOVE 'VARIABLE NO NUMÉRICA' TO W-ERROR(NUMERO-ERRORES)
            END-IF.
            IF AUX-IBAN = SPACES
               ADD 1 TO NUMERO-ERRORES
               MOVE 'DATO NO INFORMADO' TO W-ERROR(NUMERO-ERRORES)
            END-IF.
            IF AUX-RESPUESTA = SPACES
               ADD 1 TO NUMERO-ERRORES
               MOVE 'DATO NO INFORMADO' TO W-ERROR(NUMERO-ERRORES)
            END-IF.

        LLAMAR-MODULO.
```

Llamada al módulo, visualización de la llamada y visualización de errores OK

COBOL_PRINCIPAL_FXP19318.TXT: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
            MOVE 'DATO NO INFORMADO' TO W-ERROR(NUMERO-ERRORES)
        END-IF.

     LLAMAR-MODULO.
    *--------------------*
          MOVE 'FXP19318' TO MODULO
          CALL MODULO USING AUX-AREA-INTERCAMBIO.
          DISPLAY 'DATOS DE E/S AL MODULO: ' MODULO

          DISPLAY 'VALOR DE OPCION : ' AUX-OPCION.
          DISPLAY 'VALOR DE ENTIDAD : ' AUX-ENTIDAD.
          DISPLAY 'VALOR DE IBAN : ' AUX-IBAN.
          DISPLAY 'VALOR DE RESPUESTA : ' AUX-RESPUESTA.

     VISUALIZAR-ERRORES.
    *--------------------*
            DISPLAY "****************************************"
            DISPLAY "* ERROR. DATOS DE ENTRADA INCORRECTOS  **"
            DISPLAY "* REVISE ERRORES:                      **"

            MOVE 1 TO I
            PERFORM UNTIL I > NUMERO-ERRORES
                DISPLAY "* ERROR " I ": " W-ERROR(I)
                ADD 1 TO I
            END-PERFORM
            DISPLAY "****************************************"
            DISPLAY "* DATOS DE ENTRADA:                    **"
          DISPLAY 'DATOS DE E/S AL MODULO: ' MODULO

          DISPLAY 'VALOR DE OPCION : ' LNK-OPCION.
          DISPLAY 'VALOR DE ENTIDAD : ' LNK-ENTIDAD.
          DISPLAY 'VALOR DE IBAN : ' LNK-IBAN.
          DISPLAY 'VALOR DE RESPUESTA : ' LNK-RESPUESTA.
            DISPLAY "****************************************".
```

**RESULTADO OK**

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 07/05/2019 | Definición de la prueba 1h |
| | 09/05/2019 | Preparación de la prueba 2h |
| | 09/05/2019 | Finalizada la prueba |

*Tabla 86. Detalle caso de prueba CASO-U-006*

| ID: | CASO-U-007 | Responsable: | Abel |
|---|---|---|---|
| Funcionalidad/ Requisitos | **Validación sub requisito REQ-002-02.** Generación de entorno de ejecución JCL | | |

| | |
|---|---|
| **Descripción** | Verificar partes del JCL corresponden con código de entrada. |
| **Datos de entrada** | Programa cobol con 3 ficheros, 2 de entrada y uno de salida con 222 posiciones FB. (código relevante para obtener jcl)<br><br>PROGRAM-ID. FXP19318.<br>FILE-CONTROL.<br>SELECT FENTRADA ASSIGN TO FENTRADA.<br>SELECT FSALIDA ASSIGN TO FSALIDA.<br>SELECT FENTRAD2 ASSIGN TO FENTRAD2.<br>FD FENTRAD2<br>BLOCK CONTAINS 0 RECORDS<br>RECORD 0 CHARACTERS<br>DATA RECORD R-FENTRAD2.<br>01 R-FENTRAD2 PIC X(222).<br><br>OPEN INPUT FENTRADA<br>OPEN INPUT FENTRAD2<br>OPEN OUTPUT FSALIDA. |
| **Datos de salida esperados** | Generación de un JCL según ficheros y datos de entrada obtenidos del código COBOL |

**ANÁLISIS DE LOS RESULTADOS**

Selección del programa pr2 es modulo.cob (mismo código que prueba anterior)

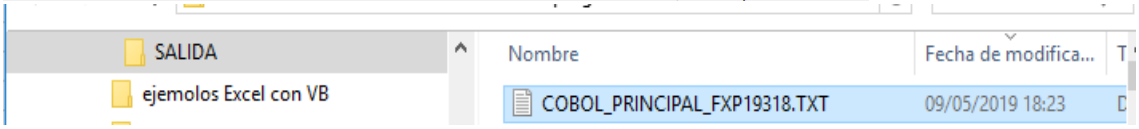
Ejecución del proceso:
Se genera JCL:

JCL_pr2 es modulo.cob: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
//JCLFXP19 JOB IT,'PRUEBAS UNITARIAS',MSGCLASS=X,CLASS=D,NOTIFY=&SYSUID,
//         RESTART=*
//*********************************************************************
//*        PRUEBAS UNITARIAS DEL PROGRAMA: FXP19318                   *
//*********************************************************************
//PASO0001 EXEC  PGM=IKJEFT1B,COND=(4,LT)
//STEPLIB  DD  DSN=BP.OBJETOS.DESARROLLO,DISP=SHR
//         DD  DSN=BP.OBJETOS.PRODUCCION,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSDBOUT DD  SYSOUT=9,HOLD=YES,FCB=S800
//SYSABOUT DD  SYSOUT=9,HOLD=YES,FCB=S800
//SYSUDUMP DD  SYSOUT=9,HOLD=YES,FCB=S800
//FENTRADA DD  DSN=TST2.KT.ES.FICHERO.FENTRADA,
//             DISP=SHR
//FENTRAD2 DD  DSN=TST2.KT.ES.FICHERO.FENTRAD2,
//             DISP=SHR
//FSALIDA DD  DSN=TST2.KT.ES.FICHERO.FSALIDA,
//             DISP=(,CATLG,DELETE),
//             SPACE=(500,(10,1),RLSE),AVGREC=K,
//             LRECL=500,RECFM=FB
//SYSTSIN  DD  *
 DSN SYSTEM(DBD1)
  RUN PROGRAM(FXP19318) PLAN(DESA)
      WHEN SYSRC(NE 0) CALL 'BP.OBJETOS.PRODUCCION(CANCELAR)'
      END
```

Línea 1, columna 1




**RESULTADO OK**

| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 07/05/2019 | Definición de la prueba 1h |
| | 09/05/2019 | Preparación de la prueba 2h |
| | 09/05/2019 | Finalizada la prueba |

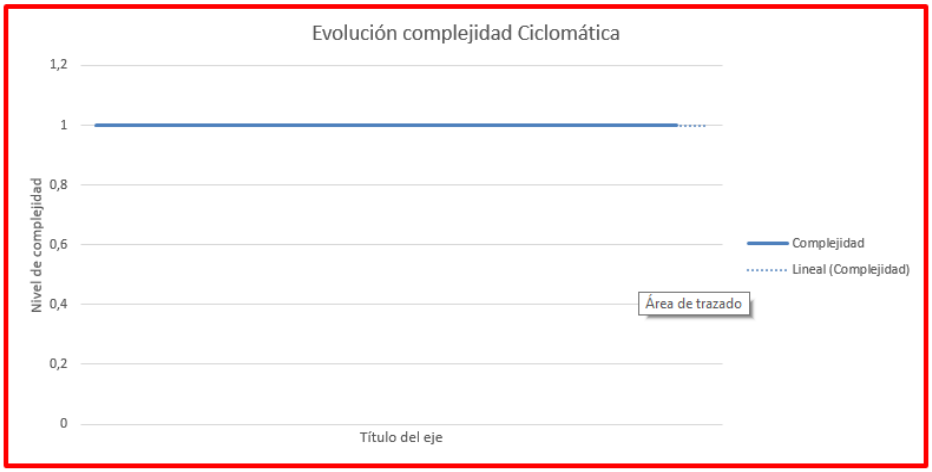
*Tabla 87. Detalle caso de prueba CASO-U-007*

| **ID:** | CASO-U-008 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad/ Requisitos** | **Validación sub requisito REQ-002-03.** Medición de la calidad y complejidad. | | |
| **Descripción** | Obtención de las siguientes métricas:<br>• Complejidad ciclomática.<br>• Métricas HALSTEAD | | |
| **Datos de entrada** | Programa cobol sencillo para poder validar fácilmente estas métrica. EL programa analizado será el pr3Operaciones.cob, ya utilizado en pruebas anteriores. | | |
| **Datos de** | Métricas mencionadas con los valores correspondientes. | | |

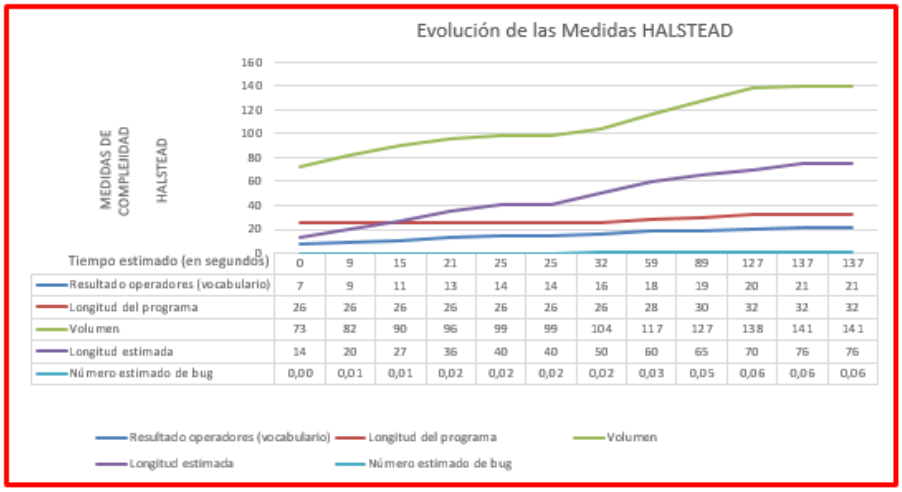
| | |
|---|---|
| **salida esperados** | Complejidad ciclomática debe resultar 1 ya que solamente contiene 1 camino.<br>En las medidas HEADTEAD contiene 16 operadores y 5 variables. |

## ANÁLISIS DE LOS RESULTADOS

Selección del programa y ejecución:
Complejidad ciclomática Ok con resultado de 1:




**Y medidas HEADTEAD ok, según resultados esperados:**




| Tiempo estimado (en segundos) | 0 | 9 | 15 | 21 | 25 | 25 | 32 | 59 | 89 | 127 | 137 | 137 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Resultado operadores (vocabulario) | 7 | 9 | 11 | 13 | 14 | 14 | 16 | 18 | 19 | 20 | 21 | 21 |
| Longitud del programa | 26 | 26 | 26 | 26 | 26 | 26 | 26 | 28 | 30 | 32 | 32 | 32 |
| Volumen | 73 | 82 | 90 | 96 | 99 | 99 | 104 | 117 | 127 | 138 | 141 | 141 |
| Longitud estimada | 14 | 20 | 27 | 36 | 40 | 40 | 50 | 60 | 65 | 70 | 76 | 76 |
| Número estimado de bug | 0,00 | 0,01 | 0,01 | 0,02 | 0,02 | 0,02 | 0,02 | 0,03 | 0,05 | 0,06 | 0,06 | 0,06 |

**MEDIDAS DE COMPLEJIDAD HALSTEAD**

| Campo | Descripción | Fórmula de cálculo | Datos entrada | Resultados |
|---|---|---|---|---|
| n1 | Número de operadores disti | Dato que se obtiene del programa | 16 | |
| n2 | Numero de variables/operar | Dato que se obtiene del programa | 5 | |
| n | Resultado operadores (voc | n=n1+n2 | | 21 |
| N1 | Total de operadores | Se obtiene del programa | 21 | |
| N2 | Total de variables/operando | Se obtiene del programa | 11 | |
| N | Longitud del programa | N=N1+N2 | | 32 |
| LE | Longitud estimada | LE = n1*log2n1+n2*log2n2 | | 76 |
| V | Volumen | V=N*log2n | | 141 |
| D | Dificultad | D=n1/2*N2/n2 | | 18 |
| E | Esfuerzo estimado | E=D*V | | 2.474 |

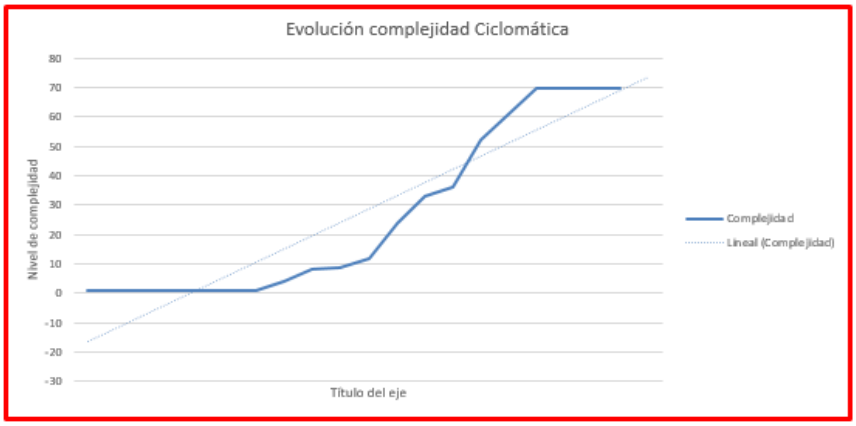
| **RESULTADO OK** | | |
|---|---|---|
| **Historia** | Fecha | Estado y duración en horas |
| | 07/05/2019 | Definición de la prueba 1h |
| | 09/05/2019 | Preparación de la prueba 2h |
| | 09/05/2019 | Finalizada la prueba |

*Tabla 88. Detalle caso de prueba CASO-U-008*

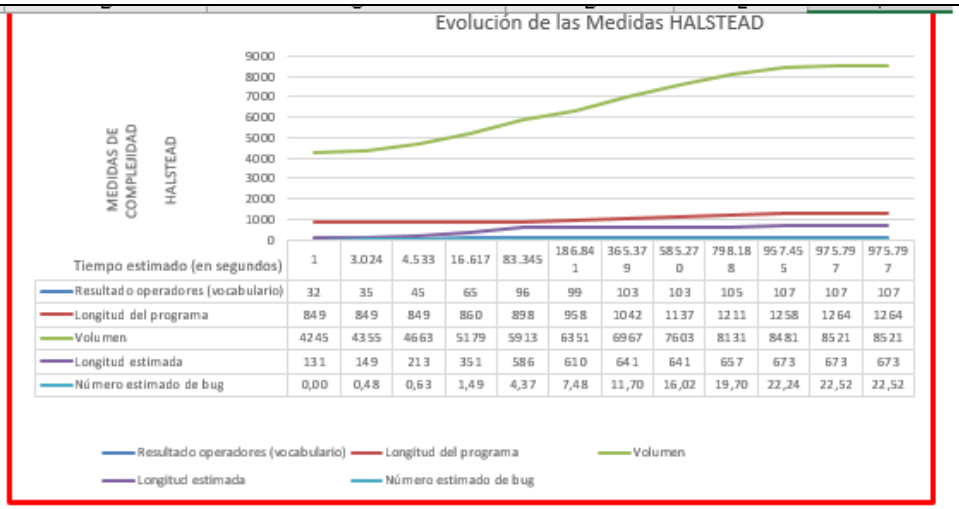
| **ID:** | CASO-U-009 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad/ Requisitos** | **Validación sub requisito REQ-002-03.** Medición de la calidad y complejidad. | | |
| **Descripción** | Obtención de las mismas métricas pero con un programa más complejo de evaluar:<br>• Complejidad ciclomática.<br>• Métricas HALSTEAD | | |
| **Datos de entrada** | Programa cobol complejo con más de 1000 líneas de codificación. | | |
| **Datos de salida esperados** | Datos correspondientes según análisis de datos de los objetos COBOL | | |

**ANÁLISIS DE LOS RESULTADOS**

Selección del programa complejo
Resultado ciclomático con 70 caminos, lo que indica que se encuentra en el nivel de complejidad más alto:




Y las medidas HEADTEAD en una de sus métricas nos indica que tiene 22 posibilidades de bug y que su codificación y pruebas se establece en unas 271h (975.797 seg)




**MEDIDAS DE COMPLEJIDAD HALSTEAD**

| Campo | Descripción | Fórmula de cálculo | Datos entrada | Resultados |
|---|---|---|---|---|
| n1 | Número de operadores disti | Dato que se obtiene del programa | 97 | |
| n2 | Numero de variables/operar | Dato que se obtiene del programa | 10 | |
| n | Resultado operadores (voc | n=n1+n2 | | 107 |
| N1 | Total de operadores | Se obtiene del programa | 839 | |
| N2 | Total de variables/operando | Se obtiene del programa | 425 | |
| N | Longitud del programa | N=N1+N2 | | 1.264 |
| LE | Longitud estimada | LE = n1*log2n1+n2*log2n2 | | 673 |
| V | Volumen | V=N*log2n | | 8.521 |
| D | Dificultad | D=n1/2*N2/n2 | | 2.061 |
| E | Esfuerzo estimado | E=D*V | | 17.564.353 |
| T | Tiempo requerido (seg) | T=E/18 | | 975.797 |
| B | Número estimado de bug | B=(E^2/3) / 3000 | | 22,523 |

**RESULTADO OK**

| Historia | Fecha | Estado y duración en horas |
|---|---|---|
| | 07/05/2019 | Definición de la prueba 1h |
| | 09/05/2019 | Preparación de la prueba 2h |
| | 09/05/2019 | Finalizada la prueba |

*Tabla 89. Detalle caso de prueba CASO-U-009*

| ID: | CASO-U-010 | Responsable: | Abel |
|---|---|---|---|
| Funcionalidad/ Requisitos | **Validación sub requisito REQ-002-04.** Detección automática de fallas o posibles problemas potenciales en el programa a través de muestreos | | |
| Descripción | Revisión de posibles errores en tiempo de ejecución según funcionalidad definida. | | |
| Datos de entrada | | | |
| Datos de salida esperados | Detección de Warning y de errores graves que pueda tener el código de entrada COBOL | | |

**ANÁLISIS DE LOS RESULTADOS**

Selección del programa y ejecución del proceso-

**RESULTADO OK**

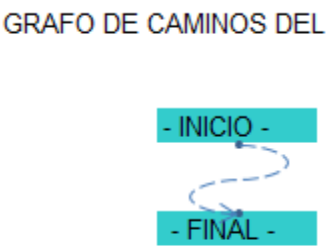
| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 08/05/2019 | Definición de la prueba 1h |
| | 08/05/2019 | Preparación de la prueba 2h |
| | 08/05/2019 | Finalizada la prueba |

*Tabla 90. Detalle caso de prueba CASO-U-010*

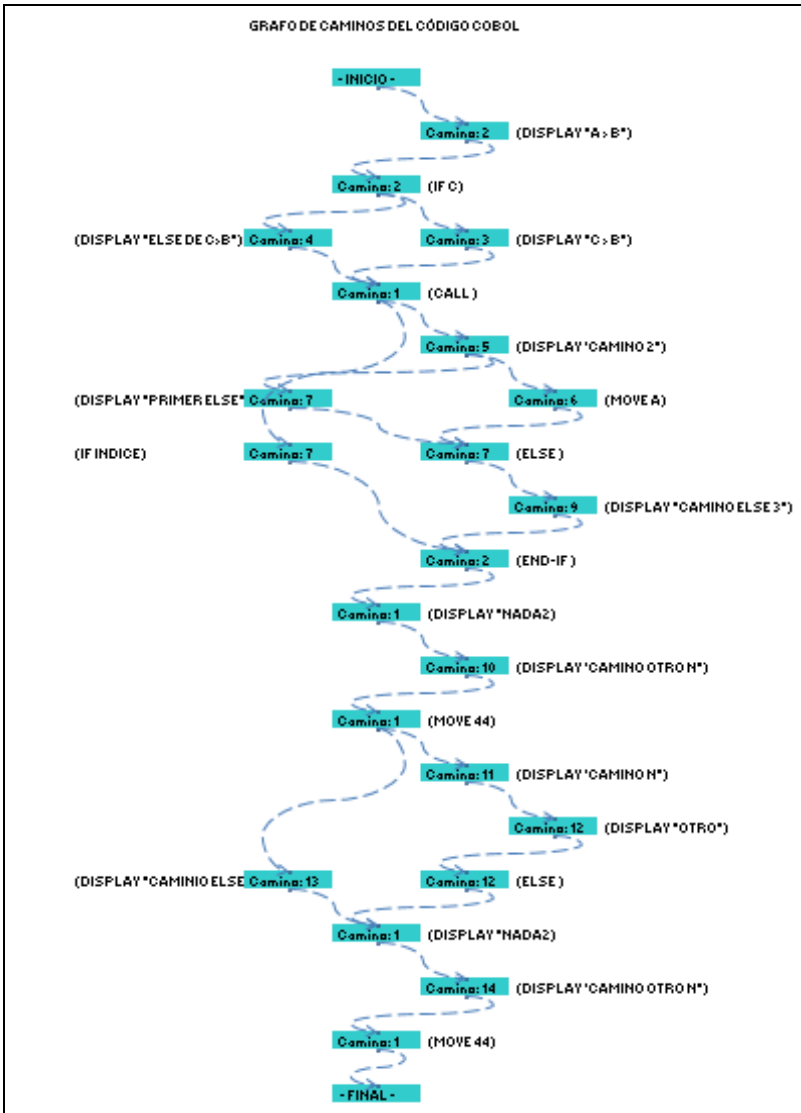
| **ID:** | CASO-U-011 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad/ Requisitos** | **Validación sub requisito REQ-002-05.** Crear un grafo con el flujo de caminos que puede tener el programa | | |
| **Descripción** | Verificar generación del diagrama de grafo de caminos. | | |
| **Datos de entrada** | Dos ejecuciones para su validación:<br>• Programa con un solo camino.<br>• Programa con 14 caminos. | | |
| **Datos de salida esperados** | Generación de diagrama de grafos según caminos del código de entrada. | | |

**ANÁLISIS DE LOS RESULTADOS**

Primer programa con un solo camino:




Programa 14 caminos:




**RESULTADO OK**

| **Historia** | Fecha | Estado y duración en horas |
|---|---|---|
| | 09/05/2019 | Definición de la prueba 1h |
| | 09/05/2019 | Preparación de la prueba 1h |
| | 09/05/2019 | Finalizada la prueba |

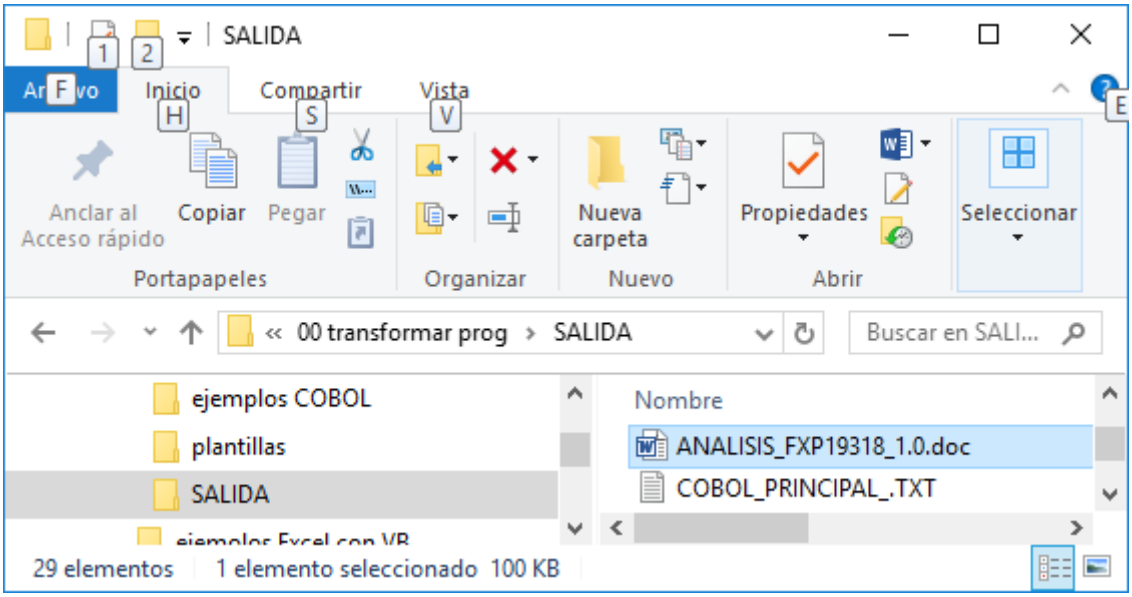
*Tabla 91. Detalle caso de prueba CASO-U-011*

| **ID:** | CASO-U-012 | **Responsable:** | Abel |
|---|---|---|---|
| **Funcionalidad/ Requisitos** | **Validación sub requisito REQ-002-06.** Generación automática del documento según formato esperado | | |
| **Descripción** | Obtención del documento según requisitos especificados. (sin entrar en contenido, solamente estructura) | | |
| **Datos de** | Programa cobol sencillo | | |

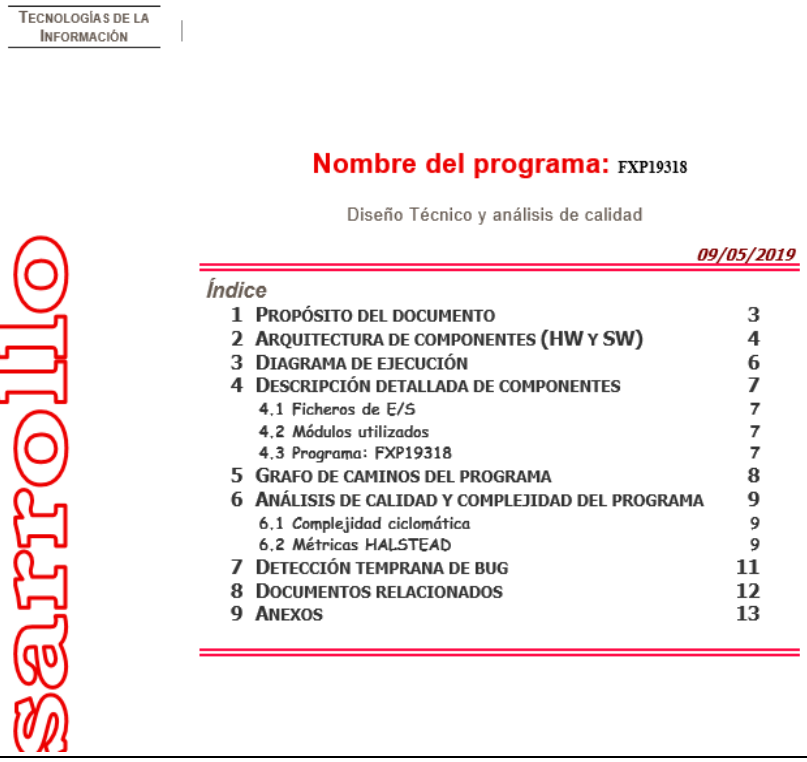
| | |
|---|---|
| **entrada** | |
| **Datos de salida esperados** | Documento WORD |

**ANÁLISIS DE LOS RESULTADOS**

Ejecución proceso
Generación del documento:




Generación de la estructura según definido en requisitos:



TECNOLOGÍAS DE LA INFORMACIÓN

**Nombre del programa:** FXP19318

Diseño Técnico y análisis de calidad

*09/05/2019*

*Índice*



sarrollo

| **RESULTADO OK** | | |
|---|---|---|
| **Historia** | Fecha | Estado y duración en horas |
| | 07/05/2019 | Definición de la prueba 1h |
| | 09/05/2019 | Preparación de la prueba 2h |
| | 09/05/2019 | Finalizada la prueba |

*Tabla 92. Detalle caso de prueba CASO-U-012*